Totally Not Instagram Release Summary

Team members

Name and Student id	GitHub id	The number of story points (or ideal hours for tasks) that a member was an **author**.
Adrien Kamran	AdrienKamran	86
Mohammad Shah Newaz	abirshah	103
Qandeel Arshad	qandeelarshad	2
Sarah Ghorbali	sarahGhrbl	0

CONTRIBUTION CHARTS

Project summary

Did you just take a picture that you want to share with the world? Do you have a photo gallery just waiting to be displayed? Totally Not Instagram was made to solve these very problems in the form of a simple Web application. By registering for a TNI account, you are able to upload and share your photos with other registered users, as well as interact through comments and likes. Each user is given their own profile gallery through which their entire catalog of uploads is displayed. On the home page, users are able to see posts made by other registered users, in reverse chronological order (newest to oldest). As such, Totally Not Instagram allows people from around the world to share their creations and stay connected.

Velocity and a list of user stories and non-story tasks for each iteration

TOTAL: 21 stories, 163 points over 12 weeks

Sprint 1: **{0 stories, 0 points, velocity: 0}**

Sprint 2: **{6 stories, 41 points, velocity : 41}**

- Non-Story Task: Create HTML, JS and CSS files
  - Points: 4 / Status: Done
- Non-Story Task: Create sign-in page
  - Points: 4 / Status: Done
- Non-Story Task: Create login page
  - Points: 6 / Status: Done
- Non-Story Task: Setting up python environment, local server
  - Points: 2 / Status: Done
- Non-Story Task: Update wiki for Han 24 + 29 meetings
  - Points: 1 / Status: Done
- User Story: “As a user, I would like to upload my picture.”
  - Points: 24 / Status: Done

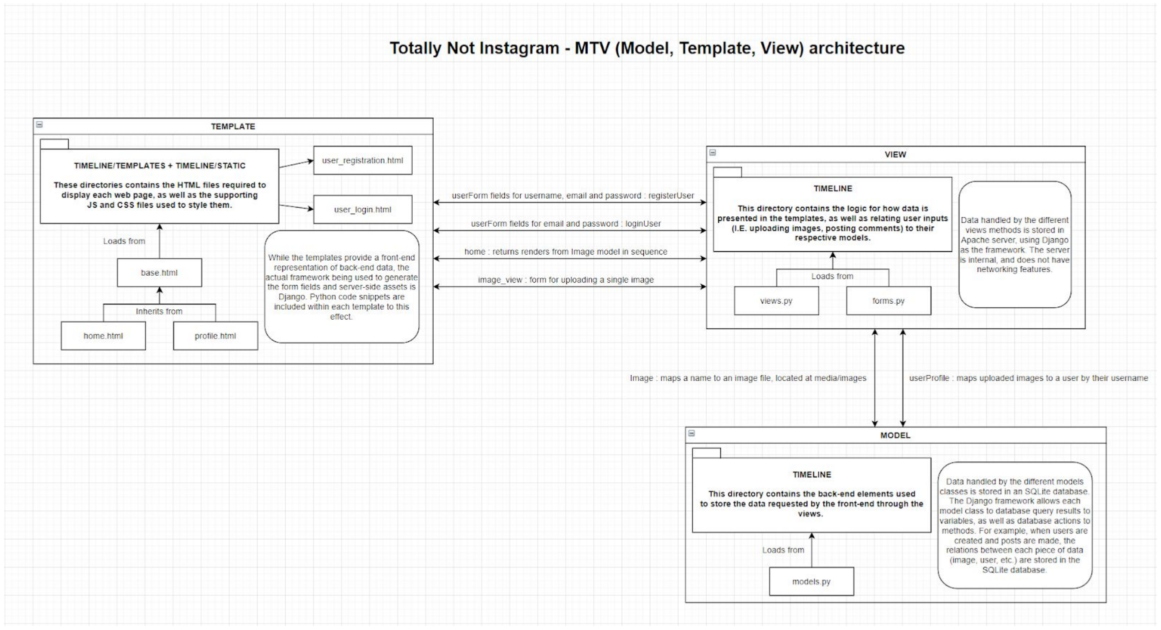
Sprint 3: **{10 stories, 90 points, velocity : 90}**

- Non-Story Task: Refactoring database code fields

    - Points: 2 / Status: Done
- Non-Story Task: User session creation
    - Points: 2 / Status: Done
- Non-Story Task: Registration back-end implementation
    - Points: 2 / Status: Done
- Non-Story Task: Login back-end implementation
    - Points: 2 / Status: Done
- Non-Story Task: Implement Travis C.I. tests using pytest
    - Points: 2 / Status: Pushed
- Non-Story Task: Create architecture diagram
    - Points: 4 / Status: Done
- User Story: “As a user, I would like to comment on a post.”
    - Points: 24 / Status: Done
- User Story: “As a user, I would like to follow other users and see their posts.”
    - Points: 24 / Status: Done
- User Story: “As a user, I would like to see my image posts from newest to oldest.”
    - Points: 4 / Status: Done
- User Story: “As a user, I would like to ‘like’ posts.”
    - Points: 24 / Status: Done

Sprint 4: {5 stories, 32 points, velocity : 32}
- Non-Story Task: Acceptance Test #1 - Uploading a picture
    - Points: 2 / Status: Done
- Non-Story Task: Acceptance Test #2 - Commenting on a post
    - Points: 2 / Status: Done
- Non-Story Task: Acceptance Test #4 - Liking a post
    - Points: 2 / Status: Done
- Non-Story Task: Acceptance Test #5 - Viewing posts from newest to oldest
    - Points: 2 / Status: Done
- Non-Story Task: Implement unit tests using Travis CI and Django’s native testing suite
    - Points: 24 / Status: Done

Overall Arch and Design


Totally Not Instagram - MTV (Model, Template, View) architecture
TEMPLATE
TIMELINE/TEMPLATES + TIMELINE/STATIC
These directories contains the HTML files required to display each web page, as well as the supporting JS and CSS files used to style them.
user_registration.html
user_login.html
Loads from
base.html
Inherits from
home.html
profile.html
While the templates provide a front-end representation of back-end data, the actual framework being used to generate the form fields and server-side assets is Django. Python code snippets are included within each template to this effect.
userForm fields for username, email and password : registerUser
userForm fields for email and password : loginUser
home : returns renders from Image model in sequence
image_view : form for uploading a single image
VIEW
TIMELINE
This directory contains the logic for how data is presented in the templates, as well as relating user inputs (I.E. uploading images, posting comments) to their respective models.
Loads from
views.py
forms.py
Data handled by the different views methods is stored in Apache server, using Django as the framework. The server is internal, and does not have networking features.
Image : maps a name to an image file, located at media/images
userProfile : maps uploaded images to a user by their username
MODEL
TIMELINE
This directory contains the back-end elements used to store the data requested by the front-end through the views.
Loads from
models.py
Data handled by the different models classes is stored in an SQLite database. The Django framework allows each model class to database query results to variables, as well as database actions to methods. For example, when users are created and posts are made, the relations between each piece of data (image, user, etc.) are stored in the SQLite database.

Infrastructure.

- **Django:**
  Django is an MTV (Model, Template, View) Web framework written primarily with Python. It provides the necessary back-and-front-end tools to develop and manage a Web application, such as user sessions and authentication, URL mapping, GET/POST for forms, and database integration.
  Two other frameworks were considered: Angular (written in JavaScript) and ASP.NET (written in C#). Django was chosen due to our collective familiarity with Python, and would allow us to begin development without learning a new language.
- **HTML:**
  To create Django-compatible templates for our Web application, the use of HTML files was necessary. It is a markup language (meaning that no scripting or dynamic code is supported within), and as such is used as the skeletal structure for the entire Web application.
  Since no other markup languages are supported by Django, our choice was clear from the start.
- **CSS:**
  CSS is a language made for visually styling HTML elements according to a “cascading” list of style rules. When formatting the layout of an HTML page, CSS enforces visual guidelines for all relevant elements.
  Alternatives for CSS aren’t common and don’t provide any significant advantages. The use of HTML in our project mandated the use of CSS by default.
- **JavaScript:**
  JavaScript (JS) is a scripting language supported by every commonly used Web browser. JS code allows for dynamic elements within an HTML-based Web application. As such, it can be used for advanced styling for the front-end and content delivery for back-end.

For HTML-based Web applications, JavaScript is the default choice with the highest level of support. Alternatives include Dart and TypeScript, but since everyone in our team had used JavaScript in the past to develop Web sites, it made sense to use it.

- **Bootstrap:**
  Bootstrap is a library/framework for the front-end of Web applications. It consists of modular HTML, CSS and JS elements which work in tandem to create a uniform style across templates. The framework requires the use of CSS, and elements are selected/styled using HTML Class tags.
  A number of alternatives exist in this line of frameworks, including Milligram and Bulma. Since Bootstrap is currently the largest of the bunch (in terms of accessibility and online resources), we chose it for our project.
- **SQLite:**
  SQLite is a relational database management system which uses SQL (Structured Query Language). It allows us to organize our Web application's data through tables and access/modify said data through queries.
  The alternative to SQLite is MySQL, which (which networked) acts as the server in a client-server infrastructure. However, since our Web application did not require a server, we chose to embed our database within the application by using SQLite.
- **PyTest:**
  PyTest is a testing framework for Python code. The complexity of the tests can surpass unit testing, which allows for the possibility of more complex acceptance tests (when referring to feature completion).
  Since Travis C.I. is required for this project, PyTest was the only compatible option for Python code.

Name Conventions

As Django is written in Python, naming conventions follow **PEP 8** where possible. As such, class names are Capitalized, method names use camelCase, and variables/arguments use lowercase.

EX:

```
class Comment(models.Model):
   user = models.ForeignKey (User,on_delete=models.CASCADE)
   img = models.ForeignKey (Image,on_delete=models.CASCADE)
   msg = models.CharField(max_length=1024)
```

Code

File path with a clickable GitHub link	Purpose
SOEN-341_Totally_Not_Instagram/TNI_Pycharm/TNI_Project/timeline/views.py	The view.py contains functions that takes a web request and return a web response
SOEN-341_Totally_Not_Instagram/TNI_P	The purpose of the models.py file is to define the structure of the stored data,

ycharm/TNI_Project/timeline/models.py	including the essential fields and its behaviors.
SOEN-341_Totally_Not_Instagram/TNI_Pycharm/TNI_Project/timeline/urls.py	The purpose of the urls.py file is to define the mapping between URLs and views.
SOEN-341_Totally_Not_Instagram/TNI_Pycharm/TNI_Project/timeline/forms.py	The purpose of the forms.py file is to create HTML forms and describes how it should appear and work.
SOEN-341_Totally_Not_Instagram/TNI_Pycharm/TNI_Project/timeline/templates/timeline/base.html	The base.html file is a basic template that is extended on every page. It is used as a frame for all the other pages of the Web Application.

Testing and Continuous Integration

Over the course of development for this web application, our Continuous Integration environment (Travis CI) became less and less reliable; unit tests which ran and completed locally would not be run remotely, making it so that none of the tests could be viewed off-machine.

As such, on our project's repository, an issue was started to document the unit tests in lieu of Travis CI. It can be viewed here: https://github.com/AdrienKamran/SOEN-341_Totally_Not_Instagram/issues/46

As for acceptance tests, we were able to perform 4 out of the 5 planned tests. This is due to how the "follow" feature was incomplete at the time of testing.

The unit test file can be found **HERE**. Screenshots for each test are linked in the table below.

(Adrien Kamran was responsible for the writing of the unit tests)

Unit Testing

Test File path with clickable GitHub link	What is it testing
Login Unit Test: authentication and redirection	Tests whether a user is able to login with a registered account, and refuse the login attempt if the credentials are wrong.
Registration Unit Test	Tests whether a user registration leads to the creation of a user in the database.
Post Creation Unit Test	Tests whether a post is created when submitting in the "Create Your Post Here!" form.
Like Unit Test	Tests whether the "like" counter is incremented when "liking" a post.

Comment Unit Test	Tests whether a comment is saved in the database once it is submitted.

Acceptance Testing

Github Issues	Which user story is it testing
Acceptance Test #1 - Uploading a picture	As a user, I would like to upload my picture.
Acceptance Test #2 - Commenting on a post	As a user, I would like to comment on a post.
Acceptance Test #4 - Liking a post	As a user, I would like to “like” posts.
Acceptance Test #5 - Viewing posts from newest to oldest	As a user, I would like to see my image posts from newest to oldest.

Describe your continuous integration environment. Include a link to your CI.
https://travis-ci.org/github/AdrienKamran/SOEN-341_Totally_Not_Instagram

As mentioned in this issue, we had a lot of trouble getting Travis CI to run the tests remotely. Travis CI aside, the project itself is stored on our GitHub repository, where each of our team members work on different features in parallel (to avoid conflicts during merges in future). Each branch is worked on locally from different machines, and once progress is made, it is saved through commits to the local branch. Upon completion of the work, the local branches are merged to the remote master branch through pull requests. Pull requests cannot be approved/merged by the developer who made the request, and as such a peer review of the code is required beforehand.

At this point, Travis CI automatically rebuilds the remote version of the repository in a Linux virtual machine. A set of requirements and behaviors are specified in a YML file and a requirements file, both of which are used by the remote virtual machine to create a recreation of the local machines. Once the build is up-to-date, the virtual machine runs the script listed at the bottom of the YML file, which is the same line of code used to run the Django tests locally.

Describe the choice of the static analysis tool and how do you run it. The static analysis tool should analyze the language that is used in the majority of your source code.

The choice of static analysis tool was made based on popularity. For Python code, Pylint is a static code linter which follows the PEP8 conventions to analyze code and give it a rating at the end.

To install it, the command line “pip install pylint” is executed. Upon installation, it is possible to run Pylint in any directory of the project; since all our written code resides in the “timeline” app of the Django project, I ran Pylint within by executing “pylint timeline”.

Attach a report as an appendix (not counted for the 6 pages) from static analysis tools by running the static analysis tool on your source code. Randomly select 10 detected problems and discuss what you see.

https://github.com/AdrienKamran/SOEN-341_Totally_Not_Instagram/blob/Adrien-1/pylint_out.txt

Pylint is extremely strict when it comes to adhering to the PEP8 style guidelines. Also, since Django has redundant systems included in the project structure by default, our Pylint final score suffered deductions for reasons outside of our control.

The vast majority of our problems have to do with line length, with a few cases of missing class docstrings for documentation purposes. However, in regards to docstrings, Django provides the documentation for all the native classes on their website, making the need for docstrings redundant.