<A Facebook-like social media platform> Release Summary

Team members

Name & Student IDs	*GitHub IDs*	*Member Workload in Percentages**
Robert Dilworth	rkd103	45%
Brett Baltz	blb820	54%
HeFeng Ou	ho170	NA
William Neeley	wtn36	NA

*: indicates a departure from the template; captures the same data but in a different format

Project summary (max one paragraph)

Chatting face-to-face and direct interaction is an essential component of establishing a good rapport with friends and family. Granted, we can all agree, excessive correspondence is cumbersome, tedious, and often a time sink. Rather, with the hustle and bustle of daily life, time is a finite and highly sought-after resource. What if you could excise a significant amount of bloat or wasted time from your schedule? Well, look no further because the website Hand-in-Hand can do just that. With just a few simple clicks, you can join an expansive online community on a platform that streamlines the monotonous and routine aspect of communal participation. Those interested in our product can become a member of the Hand-in-Hand family with "any old" device and a stable network connection. Just type in the URL http://127.0.0.1:5000 and witness our splendorous social media platform. Users can (1) post status updates; (2) share, like, edit, and comment on posts; (3) attach images or videos; (4) search and add friends; (5) customize their accounts; and (6) send direct messages to close acquaintances. We are considered "a cut above the rest" due to our reduced response times, minimalistic design, and intuitive user interface. With a heavily saturated market, we will dominate the competition by promoting our product's ease of use. Moreover, we are compensated with each and every page redirection on our site, so do not be shy. Come one, come all. "With hands clasped, we cohabitate in close association. Connect hand-in-hand- with friends near and abroad! Welcome to Hand-in-Hand!"

Velocity, List of User Stories, and Non-story Tasks for each Iteration

Total: 8 stories, 25 points over 15 weeks
Iteration 1 (0 stories, 0 points)
No User Stories Complete; this iteration primarily took care of documentation
Iteration 2 (4 stories, 15 points)
User Story A.1: Create Account [5 Points] [Status: Done]
User Story A.2: Login [5 Points] [Status: Done]
User Story A.3: Access Personal Timeline [2 Points] [Status: Done]
User Story B: Compose, Edit, and Post Statuses on Personal Timeline…
…[3 Points] [Status: Done]
Iteration 3 (2 stories, 7 points)
User Story C: Send and Accept Friend Requests; Connect with Friends…
…[4 Points] [Status: Done]

User Story D: Like, Comment, and Share Status/Timeline Posts of Friends…
…[3 Points] [Status: Done]

Iteration 4, Release (2 stories, 3 points)

Additional Feature #1: Add Support for Messaging Friends…
…[2 Points] [Status: In Progress]

Additional Feature #2: Add Support for Receiving Notifications for…
…Direct Messages [1 Point] [Status: In Progress]

Overall Arch and Design

The following labels link to the corresponding entities. For example, the "UML Class Diagram" label, when clicked, redirects to the self-same page on our group's GitHub repository. Notice how we neglected to include an "Architecture Diagram." Our stakeholder, the laboratory instructor, indicated that the documentation was not necessary.

UML Class Diagram

Block Diagram(s)

Infrastructure

Name	*Link*	*Why We Choose the Library, Framework, Database, Tool, etc.*	*Why We Neglected Alternatives*
Flask (General Web Application Framework) (Includes the following libraries: *flask, flask_login, flask_mail, flask_bcrypt, flask_wtf, wtforms, wtforms_validators, werkzeug, and flask_session*)	https://flask.palletsprojects.com/en/2.2.x/	Provided a minimalistic approach to developing and deploying a small-scale website	Similar frameworks contained excessive functionality that exceeded the scope of the project

Flask-SQLAlchemy (Database Connector; Interfaces with SQLite) (Includes the following libraries: *flask_sqlalchemy*, *sqlalchemy*)	https://flask-sqlalchemy.palletsprojects.com/en/3.0.x/	Automatically handles the handling, manipulation, creation, and usage of database objects	The selection of the library was preordained given the selected programming language and web application framework
SQLite (Database Connector; Employed by Flask-SQLAlchemy)	https://www.sqlite.org/index.html	Facilitates a database connection; used in tandem with Flask-SQLAlchemy to define tables, relationships, etc.	Flask-SQLAlchemy's documentation utilized the database connector; we did not challenge the examples found online
Pytest (Test Suite; Employed by Circle CI)	https://docs.pytest.org/en/7.2.x/	Used to define test cases automatically executed by the Continuous Integration (CI) service	Given how we selected Python as our programming language, our options were limited; other testing suites were not as readable
Circle CI (Test Suite; Interfaces with Pytest)	https://circleci.com/	Ensures that our code and the features outlined in our web application conform to our specifications and standards; retrieves and performs the test cases outlined in Pytest documents	Similar services were trapped behind a paywall, while Circle CI was free to use

Other Miscellaneous Libraries (Includes the following libraries: *rstr*—random element generation, *random*—introduces entropy into produced outputs, *base64*—encodes but does not encrypt sensitive data, *datetime—appends* timestamp to objects; *enum*—enumerates readable labels that are interpreted as integer values by the program, *os*—grants access to miscellaneous operating system interfaces, operator—used to conveniently fetch attributes)	NA	Used sparingly to accomplish/perform subtasks related to the overarching User Stories	NA

Name Conventions

Pythonic Naming Convention

Code

File Path	*Description*	*Status* *(Fully Functional or Prototype)*
.\Code\app.py	Driver code; initializes the database, loads in existing data, and runs the web application	Fully Functional
.\Code\project.db	Database storage file; houses the tables and relationships evoked in the driver code; stores data retrieved from the user while the web application is live	Fully Functional
.\test.py	Test cases; contains methods used to ensure the functionality of the features defined in the driver code	Fully Functional
.\Code\templates\personal_timeline.html	Web application asset; contains the HTML code and CSS styling used to present a user's timeline; most if not all features stem from its usage	Fully Functional

.\Code\templates\navigation_overlay.html	Web application asset; used as a template to derive other webpages; contains the navigational widgets such as the account page, timeline page, etc. in addition to the user search bar	Fully Functional

Testing and Continuous Integration

Each story needs a test before it is complete. If some class/methods are missing unit tests, please describe why and how you are checking their quality. Please describe any unusual/unique aspects of your testing approach.

List the **5** most important unit test with links below.

File Path	*Description*
.\test.py	*test_valid_user_login_and_logout()*: tests the website's login/logout procedure
	test_valid_credential_retrival(): tests the website's ability to locate a user's email and send a new, randomly generated password
	test_valid_user_post_and_timeline_visibility(): tests the website's ability to display a user's timeline in addition to whether or not the web application can facilitate the posting of statuses
	test_valid_status_deletion_and_editing(): tests whether or not a user can edit or delete an existing post
	test_valid_media_attachment(): tests the user's ability to attach images or video to posts
	Missing Test: **Notice how the create account routine was not tested; the procedure has built-in error handling rendering the testing of the feature redundant.**

List the **5** most important acceptance tests with links below.

File Path	*Description*
.\test.py	*test_sending_friend_request_and_verify_obfuscated_foreign_user_content():* ensures that a user can send a friend request to another user
	test_sent_friend_request_account_changes_accpeting_friend_request_and_removing_friend(): ensures that the correct

	account changes occur when a user accepts a friend request
	test_valid_post_like(): ensures that a user can like another user's post
	test_valid_post_share(): ensures that a user can share another user's post
	test_valid_post_commenting(): ensures that a user can comment on another user's post
	Missing Test: **Notice how the user's account customization was not tested; this user action can be manually tested by accessing the appropriate section of the settings tab**

Describe your continuous integration environment.

Link:
Continuous Integration Service

Description:
Our CI service, for all intents and purposes, is a standard implementation. It (A) sets up a virtual environment in a Linux operating system, (B) retrieves the code tied to our repository, (C) installs dependencies with pip using the project's requirements.txt file, and (E) executes the tests contained within the test.py document.

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.

Static Analysis Tool:
Given how most of the code was written in Visual Studio Code, we heavily relied upon the text editor's inherit linting feature. Rather, we did not explicitly deploy a static analysis tool such as Pylint. We simply addressed each syntactical and stylistic issue as they were pointed out by the "problems" terminal tab.

Static Analysis Report:
At present, Visual Studio Code has indicated that our source code is without error, which means that there are no issues to describe or discuss.