**Program:**

When researching “backend Web programs” for practice ideas, one such idea was to mimic a website “log in” screen. This allows the user to enter an email address and a password. This information is checked against data from a text file that is stored in a hash map. The program also allows a new “account” to be created.

**Goals:**

* To get experience programming a “real-world” solution
* To get more familiarity with program planning before any coding takes place
* To create effective and efficient code
* To apply test-based development

**Approach:**

I started with a “top-down” approach, creating a flowchart of logic for the program to follow. Using this information, I determined the modules that needed to be created. Then coming from a “bottom-up” approach, I designed each module including names, data structures, and functions/methods.

I coded the methods, then used the flowchart to stitch them together into a functional program.

**Outcome:**

The “bottom-up” portion of determining modules and their characteristics worked very well. These were coded first, along with their tests.

The “top-down” portion of flowchart logic however was weak, based on the following observations:

* The flowchart contained some unnecessary steps, such as “checking if a file opened successfully” and “data successfully read from file”, which in Rust happens at the same time.
* The flowchart missed some important steps, such as validating email address input.
* When it came time to “stitch together” the modules, the logic was not as straight-forward as it seemed. This resulted in messy and “on the fly” code that, while working, was inelegant. Since the flowchart was a “high level” view of the logic, it did not take certain logical nuances into account. It would be a good idea to develop the “top-down” logic more thoroughly in the future.
* This also created a few scenarios with unforeseen requirements from the modules, which required additional functionality to be added later on in the coding process.

**Improvements:**

* Plan the top-level logic more carefully; include more detail on the “stitching” logic.
* Spend more time review the design before coding.
* Create “top level logic” expectations, which can help fill out modules and determine glue code.