REPORT 3:

GOsling

Team:

Zhunussova Meruyert 200103534
Bekmagambetova Diana 200103329

Task Division

To ensure a smooth and efficient development process, we divided the tasks among ourselves based on our areas of expertise.

Diana is responsible for:

- Used Go's html/template package to render HTML templates. This allowed to easily display dynamic data on the web pages. The html/template package also ensured that the data passed to the templates was type-safe, which reduced the likelihood of bugs and errors.
- Created a template cache: implemented a template cache, which stores the rendered HTML templates in memory. This avoids the need to read the templates from disk for each HTTP request. This optimization can greatly improve the performance of the application.
- Handled template rendering errors gracefully at runtime, was added error handling to the code, so if there are any issues with rendering the templates, the application will not crash. Instead, the error is caught and a user-friendly message is displayed to the user.
- created a template function that can be called within any template to retrieve and display common dynamic data, such as the website's name or logo. This allowed to avoid duplicating code across multiple templates, which makes the code more efficient and easier to maintain.

Overall, these steps improved the performance, usability, and maintainability of the travel ticket selling application.

Meruyert is responsible for:

- Added custom functions to format and display data in the templates, improving their readability and visual appeal.
- Created middleware to set useful security headers on every HTTP response. The X-Frame-Options header set to "deny" prevents the website from being embedded within an iframe, which can protect against clickjacking attacks. The X-XSS-Protection header set to "1; mode=block" enables the browser's built-in XSS protection, which can protect against cross-site scripting attacks. By adding these security headers, was improved the overall security of the application and helped to protect users from potential security vulnerabilities.
- As part of my work on the travel ticket selling application, was implemented RESTful routes. This allowed to organize code and endpoints in a way that followed best practices for RESTful architecture. By using HTTP methods such as GET, POST, PUT, and DELETE in a consistent and standardized way.
- Parsing and accessing form data sent in a POST request: To allow users to submit data through forms on the website, implemented the ability to parse and access form data sent in a POST request. This allowed to process user input and create new data in the database based on the form data submitted by the user.
- Techniques for performing common validation checks on form data: To ensure that user input was valid and consistent with the requirements of the application, was implemented techniques for performing common validation checks on the form data. For example, checking that required fields were not empty, that email

addresses were in a valid format, and that passwords met certain criteria.

- Implemented a user-friendly pattern for alerting the user to validation failures and re-populating form fields with previously submitted data. This allowed users to easily correct any mistakes they made and submit the form again without having to re-enter all of the data.

- To make validation more scalable and reusable across the application, was created a form helper in a separate reusable package. This allowed to keep handlers clean and free of repetitive validation code, while also allowing me to easily add new validation checks and techniques in the future.

In conclusion, the additions made to the travel ticket selling application improved its functionality, security, and user experience. The implementation of RESTful routes, form data parsing, validation checks, custom functions, and security headers enhanced the application's overall robustness and efficiency. The creation of a reusable form helper package also made it easier to maintain and update the application in the future.

Future work:
Will be implemented user registration and login functionality by creating forms at /user/signup and /user/login, respectively. User data will be stored in a new users database table. We will check the user's credentials against this table and add their ID to the session data upon successful login. We will check for the "userID" value in subsequent requests to determine if the user is logged in or not.

Conclusion

Our progress on the GOsling project can be tracked on our GitHub repository at https://github.com/mistledi/GOsling.