DOCUMENTATION OF VACATION TRACKING APPLICATION

By

Gabriela Yordanova Vasileva,
Faculty number: 213110004

A Term Paper for University Internship

Major: Computing and Computer Science

Under the Supervision of
Professor Antonina Ivanova

Varna, Bulgaria
February, 2022

Contents

Introduction

The documentation describes the functionality of the “Vacation Tracker” application from both user and developer perspectives and is divided respectively in two parts which can be read independently according to the role and needs of the reader.
The entire code of the application can be viewed at:
https://github.com/GabrielaVasileva/Vacation_02

Motivation

The “Vacation Tracker” application serves to fill a gap in the current administrative processing of personal holidays of the employees of the Varna Free University "Chernorizets Hrabar" (VFU). Currently the vacation requests are manually submitted, approved by manager and forwarded to the administration. There is not an easy way to quickly visualize how many days an employee already had taken off and how many days remain. In addition, it is not possible for the employee to check the status of the vacation request. The proposed application aims to provide a solution by deliverying an online application for electronic processing of employee personal holidays.

Functionality

The following functionality is already implemented in the application:

- User logging and registration
- Vacation request submission
- Tracking of the vacation requests
- Overview of past and current vacations

Further functionality that is planned to be implemented in the future:

- Role based separation of the vacation tracking information:
    - Administrative board
    - Employee board
- Email notification
- Authorization of users through email
- Calculation of days left per user
- Automatic work days calculation within the request form from the start and end date
- Personal Holiday suggestions according to national bank holidays

User Guide

The Vacation Tracker app goals are to make requesting vacations easier and to collect all the information about vacations in a single place.

To start using the application, the user needs to be registered and then authorized when entering the system.

Registration

New users can create an account through the registration form. After filling the fields with valid data, the user needs to click the **Create Account** button and will be automatically redirected to the vacation page.

If the user tries to register with an email that already exists in the user database, a message will appear in the header of the page below the navigation bar: “Email Address already exists! Please try a different email address.”

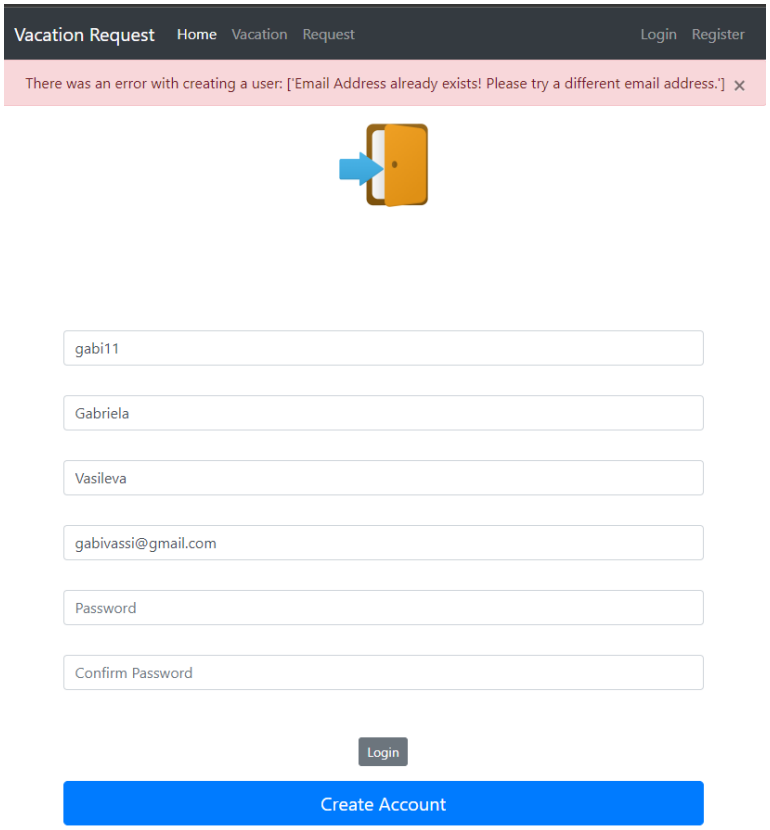

Figure 1: Registration Form and Error Notification for Existing Email in the Database

If the user tries to register with an invalid email address, a message will appear: “Invalid email address.”

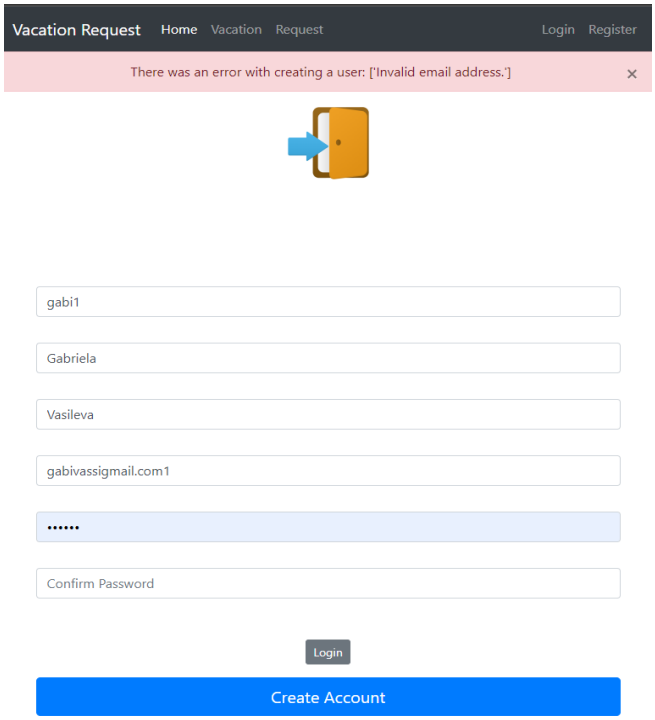

Figure 2: Registration Form and Error Notification for Invalid Email Address

If the user tries to register with a username that already exists in the user database, a notification will be shown in the bar below the navigation: “Username already exists! Please try a different username.”

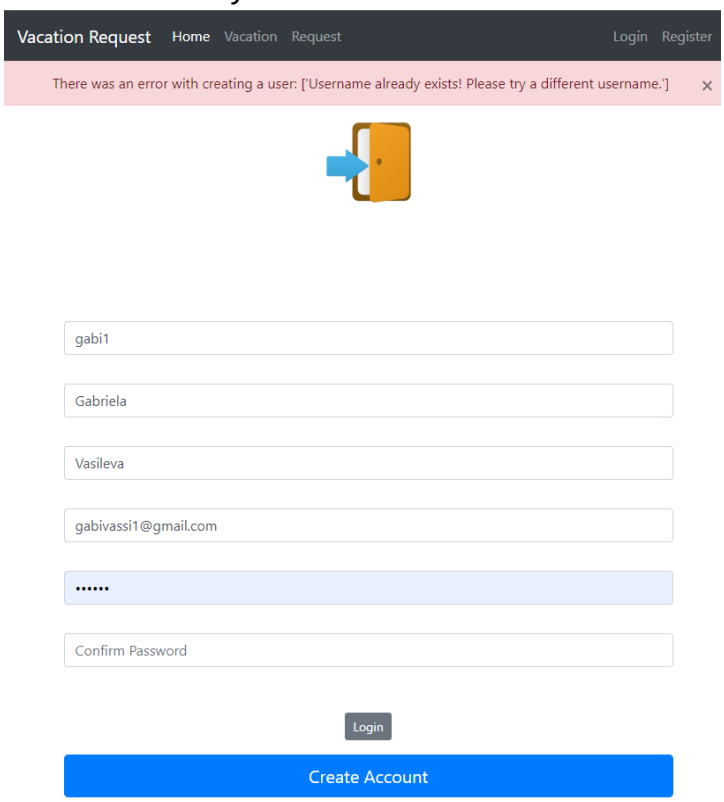

Figure 3: Invalid User Name Error Message on Registration Form

If the entries in the **Password** and **Confirm Password** fields do not match, the user will receive a message: ”Confirm Password must be equal to password!”

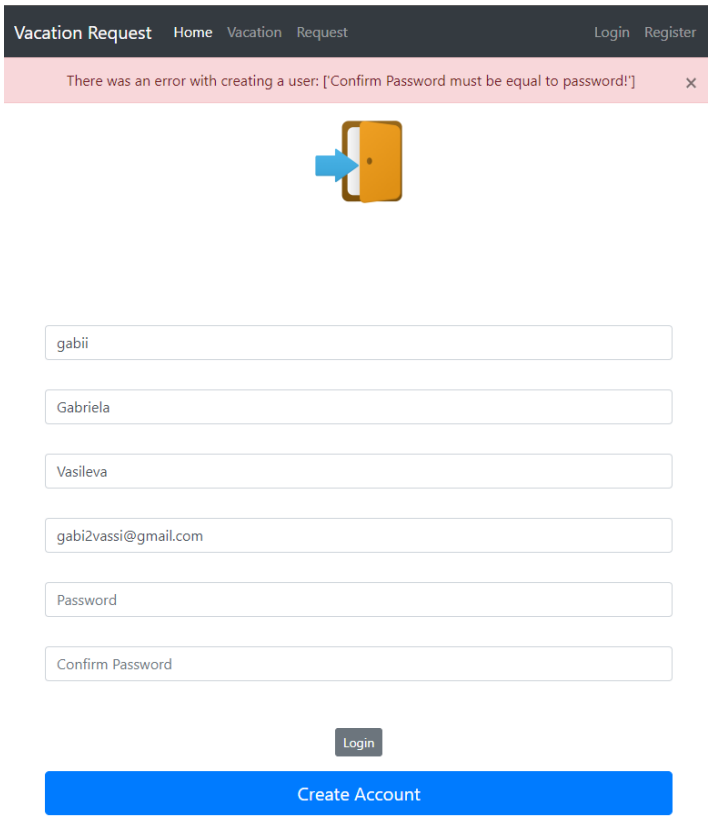

Figure 4: Registration Form Error Message for not Matching Confirming Password

The length of the **User Name**, **First Name** and **Last Name** fields must be between 2 and 30 characters, otherwise the user will receive an error message.

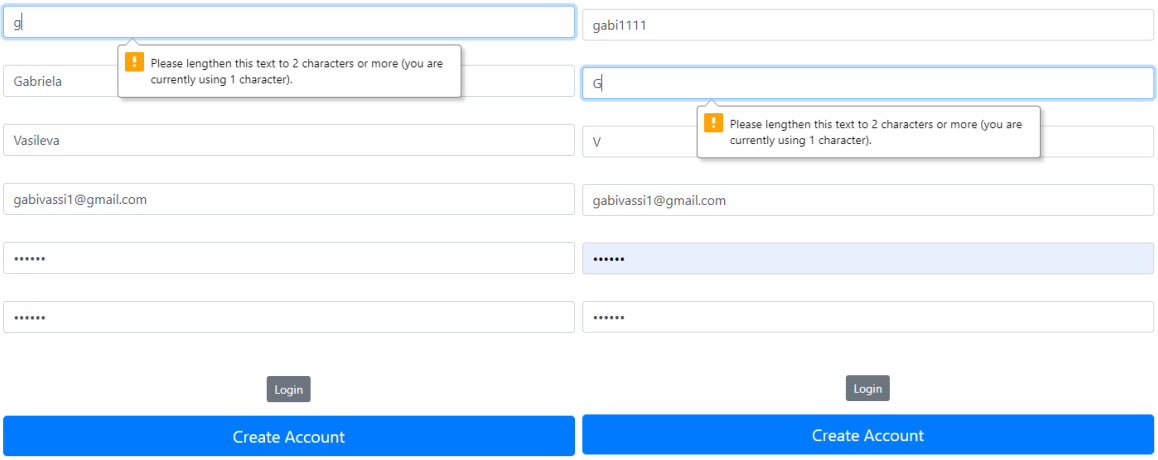

Figure 5: Minimum Character Notification for Username, First Name and Last Name fields

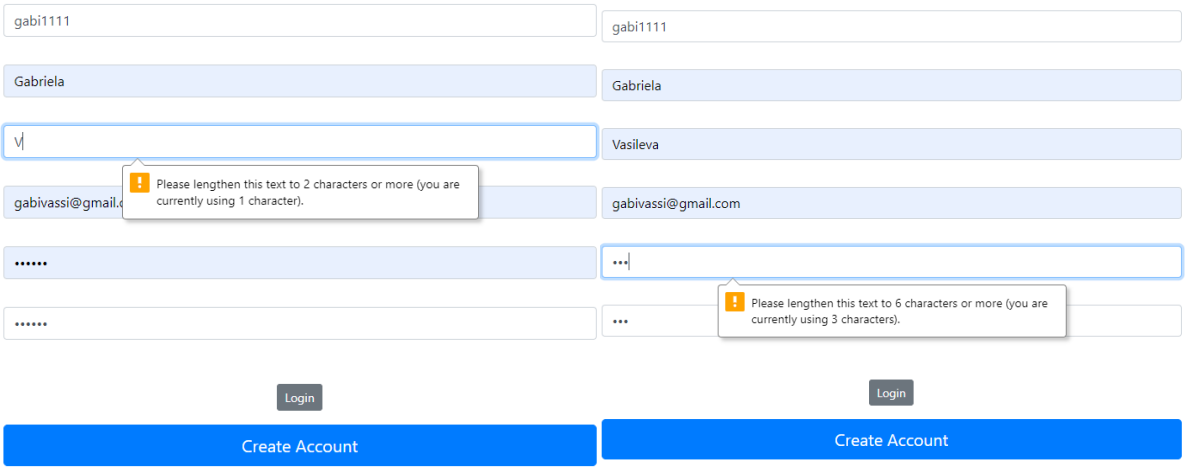

Figure 6: Error Message for Password Length Vallidation

Passwords must be more than six characters, otherwise, the user will receive an error message.

On successful registration, the user data is recorded in the database and it is possible to login to the system

Login

Once the user has a successful registration, it is possible to enter the application through the **Login** page with username and password. After successful login, the user is automatically redirected to the **Vacation** page.

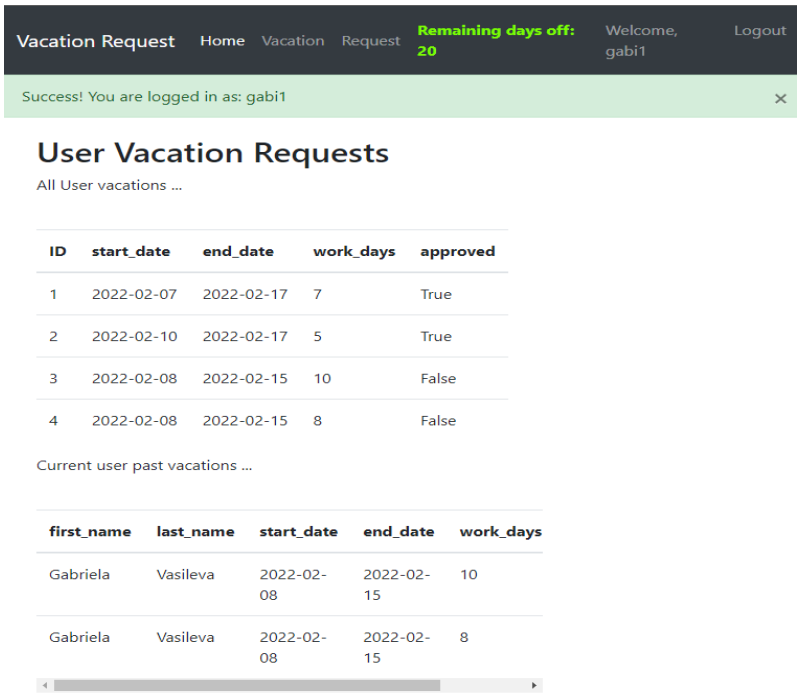

Figure 7: Vacation Page

If user enters a wrong password, an error message will show up: 'Username and password are not match! Please try again'.

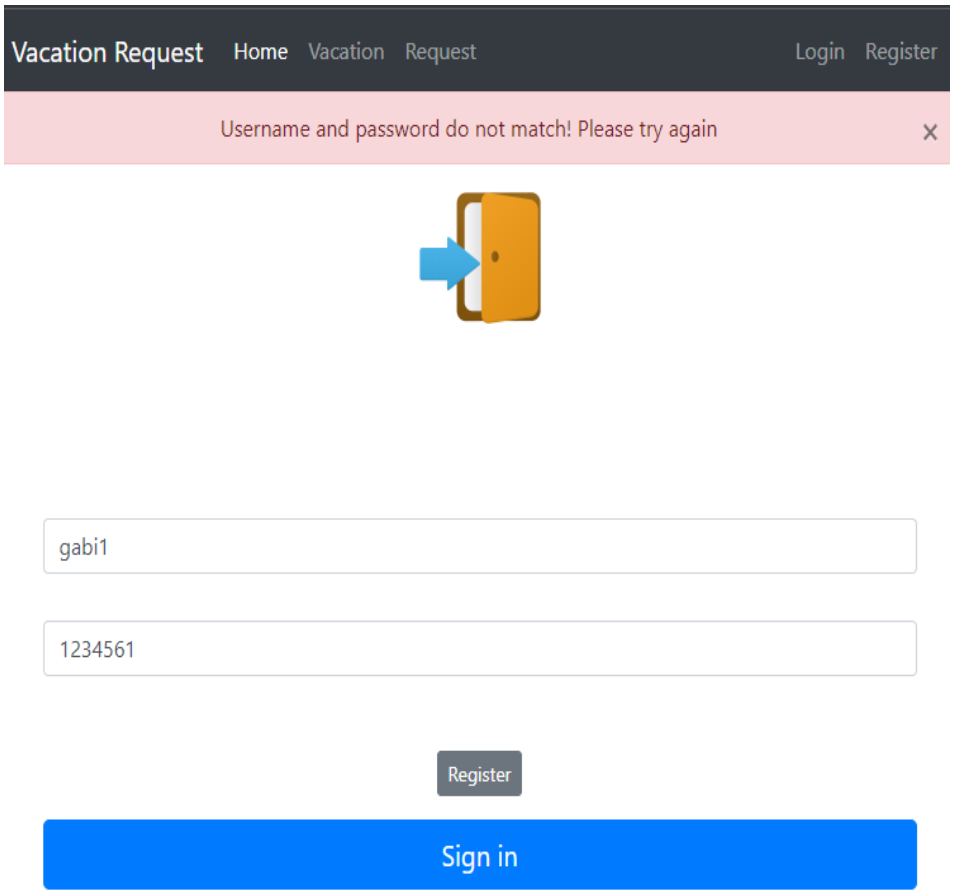

Figure 8: Login Form Error Message for Incorrect Username or Password

Vacation page

The Vacation page provides view of all user vacation requests, as well as the requests of the logged-in user. In the future, this view will be different according to the role user. The administrators and managers will see all user vacation requests. The employees will see only their own requests and status.

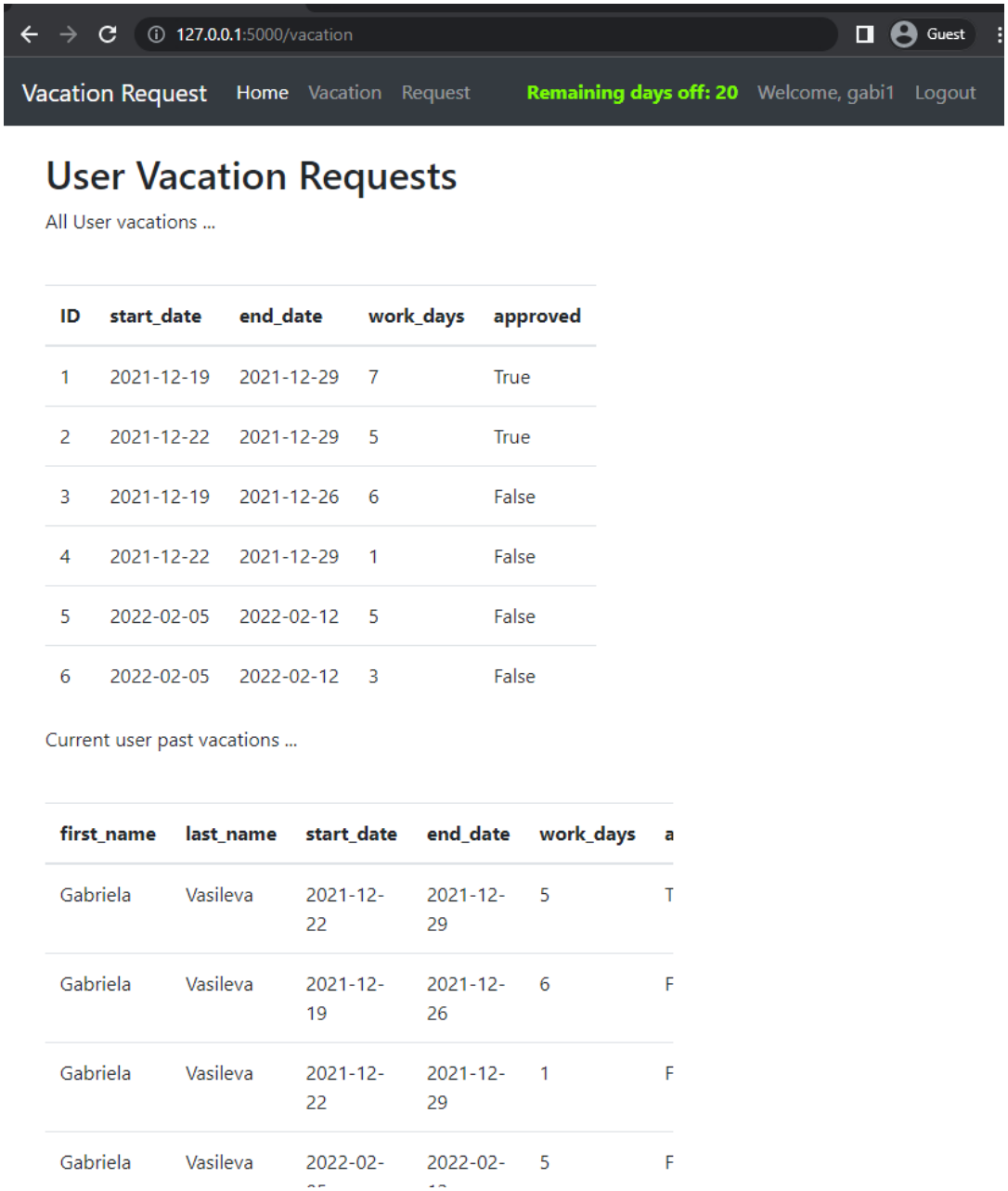

Figure 9: Vacation Page

On the vacation page there are two tables. The first one represents the vacations for all users. The second one represents logged user vacations.

Vacation request

The **Vacation Request** page provides a form for submitting individual holiday applications. There are fields for start and end date, the count of the working days, reason for the vacation, and comments. The dates are filled through calendar popups.

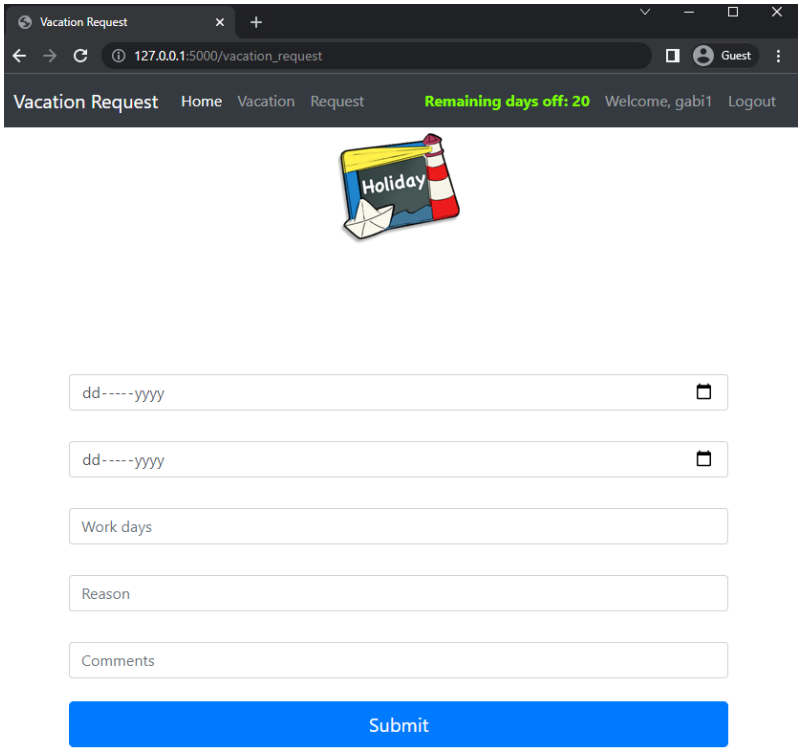

Figure 10: Vacation Request Form

When the user enters the data, and clicks on the **Submit** button. The information is recorded in the database and then visualized on the **Vacation** page. The individual vacation can approved by the manager.

Developer Documentation

Software architecture

The vacation tracker software has several tiers. A tier is a logical separation of components in an application. This separation is at a component level, not the code level. The components are database, backend application server, user interface, messaging, caching, all running in conjunction to form an online service.
The web service consists of the following architectural components.

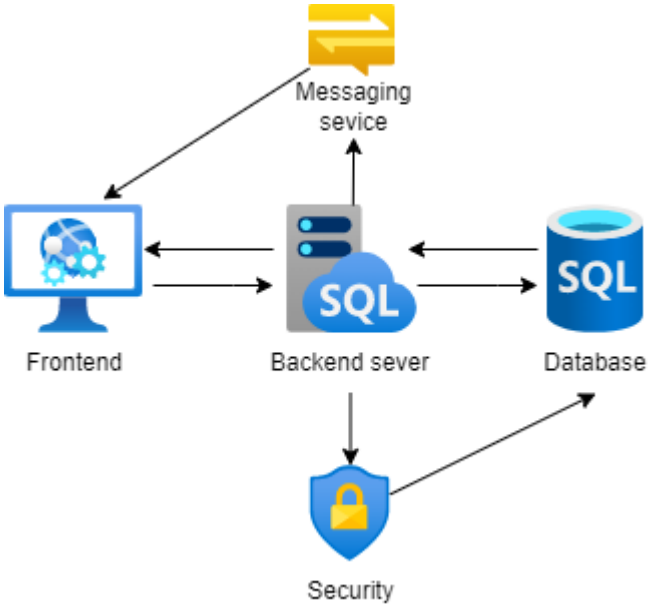

Figure 11: Three-tier Architecture

This is a three-tier application. During production, the backend, database, and frontend reside on different machines. The user interface, business logic, and the database are physically separated. Three-tier applications are popular and largely used on the web.

The user interface is written using HTML, JavaScript, CSS, and Bootstrap. The backend logic runs on a Flask server. The database lies on SQLAlchemy with SQLite client.

The client-server architecture is the fundamental building block of the Vacation Tracker. It is based on a request-response model.

The client sends a request to the server, and the server responds with the needed information.

User interface

The client holds the user interface. It is the representation part. It is developed with HTML, JavaScript, CSS and is responsible for the look and feel of the software. The user interface runs on the client, which is the gateway to our app.

The client is a browser, running commands to interact with the backend server. The client sends a request to the server to register a personal holiday, and the server persists the information in the database.

Technologies for Client Implementation

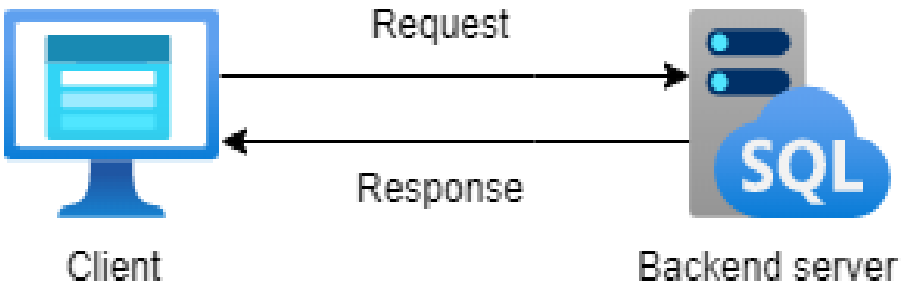

Figure 12: Client - Server Communication

There are a plethora of technologies that can be leveraged for writing the front-end. In our case, we are using the Bootstrap library, along with HTML, JavaScript, and CSS. My choice is based on most popular programming, scripting and markup languages according to the latest developer survey run by StackOverflow.

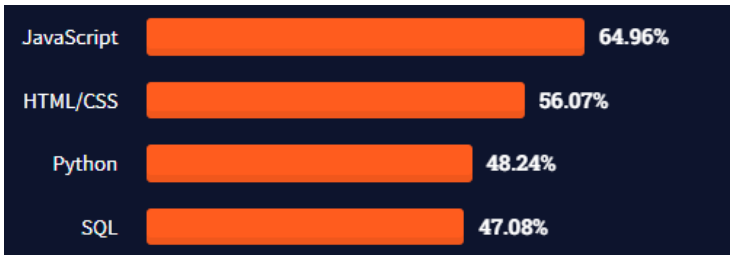

Figure 13: Most Popular Programming Languages

source: https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies

JavaScript is considered the most useful programming language for front-end development by 64.96% of the survey participants. HTML and CSS are an integral part of every website. It is curious that Python traded places with SQL and became the third most popular programming language.

Our client is a so-called thin client. The thin client holds just the user interface of the application. It contains no business logic of any sort. For every action, the client sends a request to the backend server.

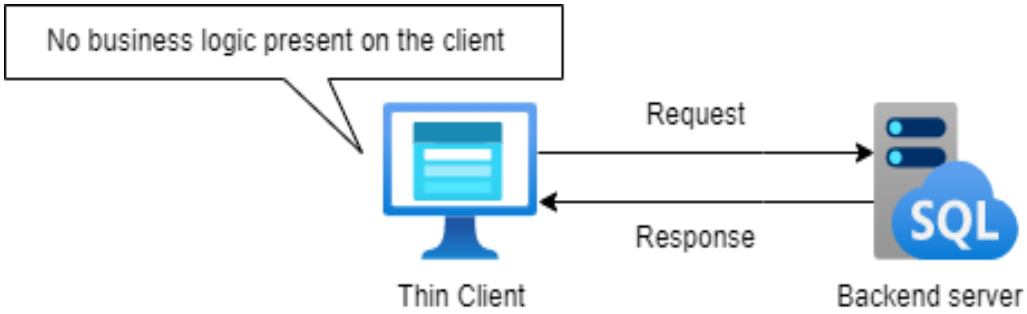

Figure 14: Client Server Decoupling

Web Server

The web server receives the requests from the client and provides the response after executing the business logic based on the request parameters received from the client. All the components of a web application need a server to run. We use a server to render the user interface on the backend and then send the generated data to the client. This is called server-side rendering.

Communication between the client and the server occurs in the form of a request-response model. The client sends the request and the server responds with the data. The entire communication happens over the HTTP protocol. The HTTP (Hypertext Transfer Protocol) is an application protocol for distributed, collaborative, hypermedia information systems, that allow users to communicate data on the World Wide Web.

The backend code implements REST-API. This acts as an interface to the outside world requests.

REST stands for Representational State Transfer. Web services that are implemented through the REST architectural style are known as RESTful web services.

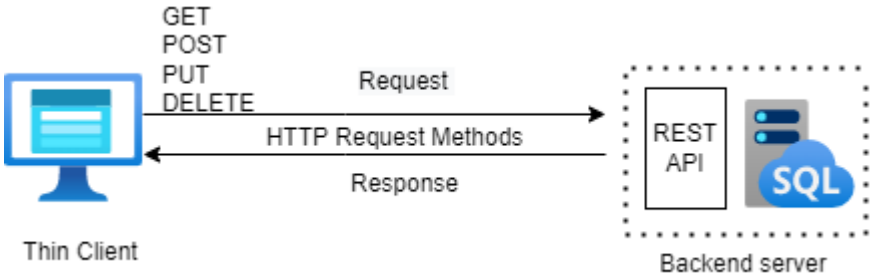

Figure 15: Representational State Transfer between Client and Server

The interaction between the client and the server is a stateless process. That means that every communication between the client and the server is like a new one. The state of the previous transmission does not affect the next. Each interchange is independent. No information or memory is carried over from past events. That is why the client needs to send the authentication information to the server every time, which enables the backend to figure out whether the client is authorized to access data. Thanks to the REST API the backend and the client code are decoupled.

The client pulls the data from the server whenever requested. For every response, there has to be a request first. The client sends the request and the server responds with the data. This is the default mode of HTTP communication, called the HTTP PULL mechanism. Every interaction between the client and the server consumes bandwidth. Every hit on the server costs the business money and adds to the load on the server.

HTTP Pull Mechanism

In the application fetching data from the server is done by sending an HTTP GET request to the server by manually triggering an event on the user interface. For example, by clicking a button or interacting with any other element on the web page. In the regular client-server communication, which is HTTP PULL, there is a Time to Live(TTL) for every request. It could be 30 seconds to 60 seconds, depending on the browser. If the client does not receive a response from the server within the TTL, the browser kills the connection, and the client has to re-send the request hoping the data from the server before the TTL ends again.

Open connections consume resources, and there is a limit to the number of open connections a server can handle at one point. If the connections do not close and new ones are introduced regularly over time, the server will run out of memory. Hence, the TTL is used in client-server communication.

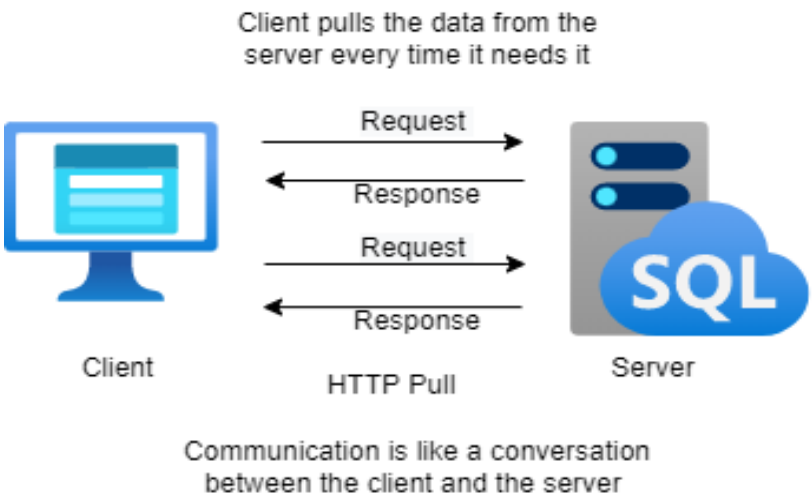

Figure 16: Client Server HTTP GET and HTTP Pull

HTTP Status Codes

Responses can include one of the following HTTP status codes, in addition to content data. Status codes are not part of endpoint descriptions because the implementation follows the HTTP standard for reporting status. But, the documentation notes status codes with special significance for the endpoint, or whose software specific cause differs from the standard.

See the Hypertext Transfer Protocol -- HTTP/1.1, Status Code Definitions standard for the complete list of status codes and their meaning.

Status code	Generalized description
200	Request completed successfully.
201	Create request completed successfully.
400	Request error. See response body for details.
401	Authentication failure, invalid access credentials.
402	In-use Splunk Enterprise license disables this feature.
403	Insufficient permission.
404	Requested endpoint does not exist.
409	Invalid operation for this endpoint. See response body for details.
500	Unspecified internal server error. See response body for details.
503	Feature is disabled in configuration file.

REST endpoints

An API/REST/Backend endpoint means the URL of the service that the client hit. With the availability of the endpoint, the backend service does not have to worry about client implementation. Any client with the required authorization to access a resource can access it.

After REST API implementation backend, the client type does not matter. We need to provide the service endpoints to the client, and it will receive the response in a standard data transport format. It is now the responsibility of the client to parse and render the response data. With REST, we can introduce any number of new clients without having to worry about the backend implementation.

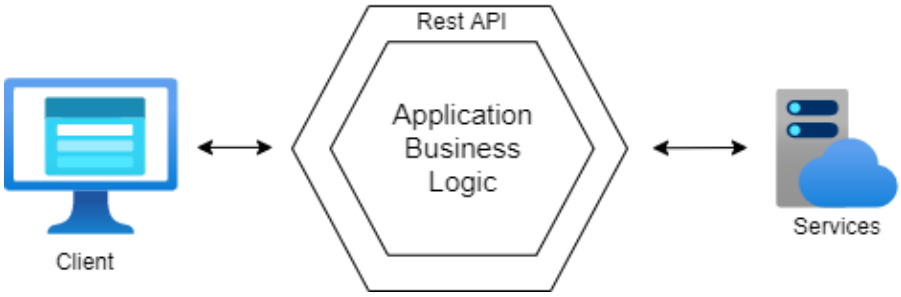

Figure 17: REST API for Application Business Logic

The REST-API acts as a gateway or a single-entry point into the system. It encapsulates the business logic and handles all the client requests the input data, and other necessary tasks before providing access to the application resources.

Login

The /login endpoint signs the user in. It loads the login page and presents the authentication options configured for the client to the user.
The /login endpoint only supports HTTPS GET. The user client makes this request through a browser. It handles the common tasks of logging in, logging out, and remembering users' sessions
GET /login
Loads in the browser the login page.

POST /login
Sends username and password for basic authorization and redirects to the vacation page if the authorization is successful.

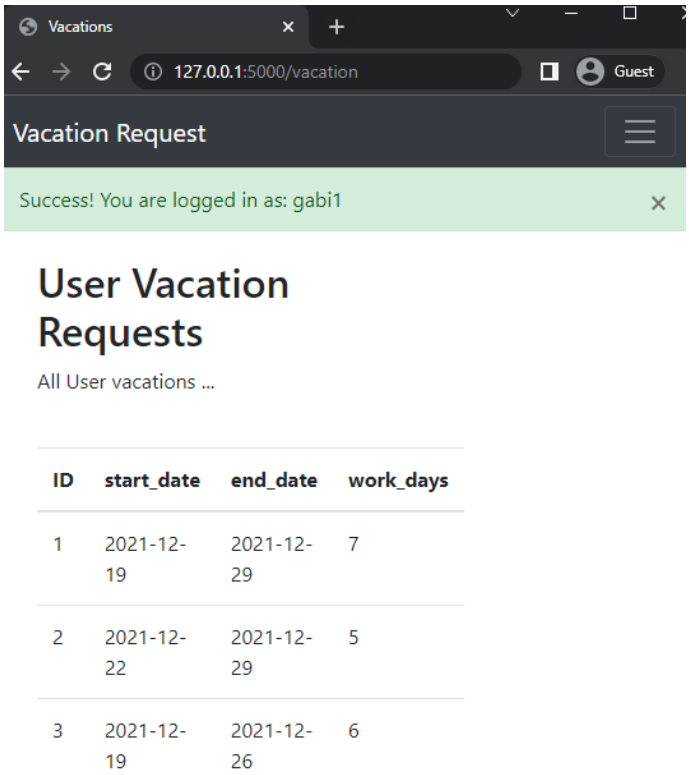

Figure 18: Vacation Endpoint

Otherwise, the user gets a message, "Username and password are not match! Please try again"

If the method is POST and the form is valid, it searches the user in database using the username. If the user exists, it compares the hashed password which is stored inside the database and the simple password which is entered by the user.

If both password match, it allows the user to access and redirect to the vacation page while saving username and email inside the session.

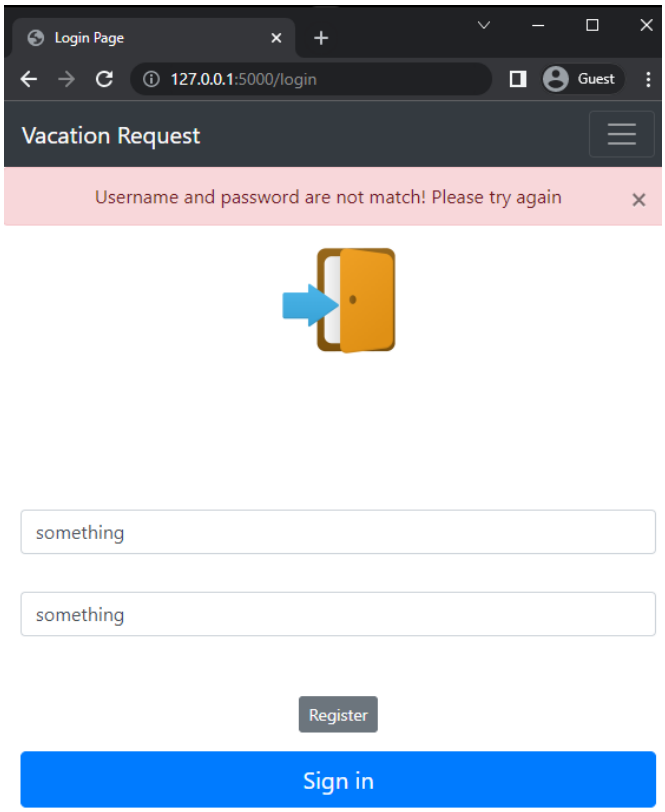

Figure 19: Logging Form Validation

Register

The register endpoint has GET and POST methods. If the HTTP request method is GET, then the registration form is rendered. The registration form contains fields for username, first name, last name, email address, password, and password confirmation.

When the form is filled and the create account button is clicked, the POST request is triggered.

With the HTTP POST request, the form is validated for email format, duplicate username and password criteria. If the username is present in the database, a message is displayed: 'Username already exists! Please try a different username.' If the email address already exist, there is a notification: 'Email Address already exists! Please try a different email address.' In case the form is valid, it generates hash password from the user password and creates a new user object with that password. It saves the user object to the user table. There is a notification: "Account created successfully! You are now logged in as {user_to_create.username}".

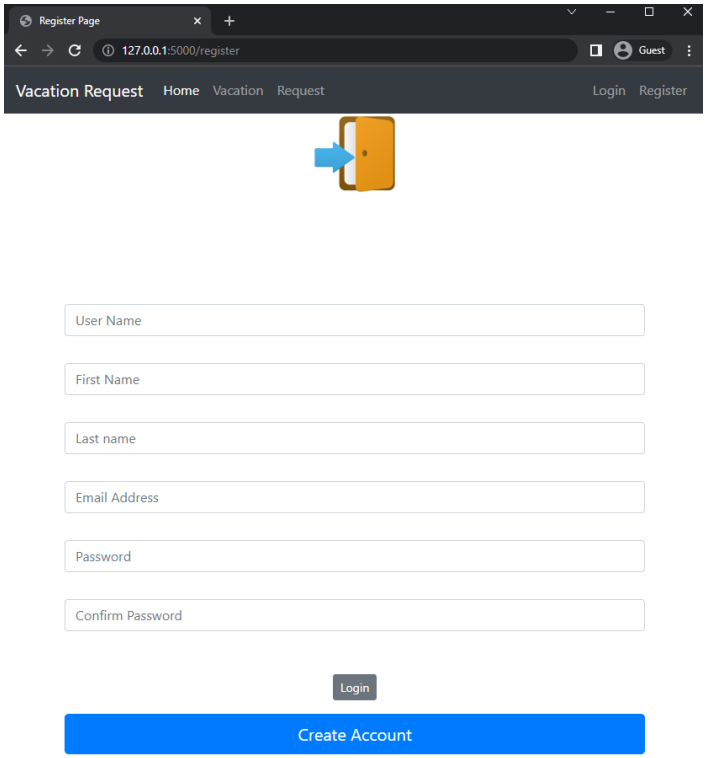

Figure 20: Registration Endpoint

Register user page REST endpoint: /register

Vacation Request

The GET method of the /vacation_request endpoint renders the vacation form. The form consists of the following fields – start date of vacation, end date of the vacation, number of work days, reason for the vacation, and an optional field for commenter.

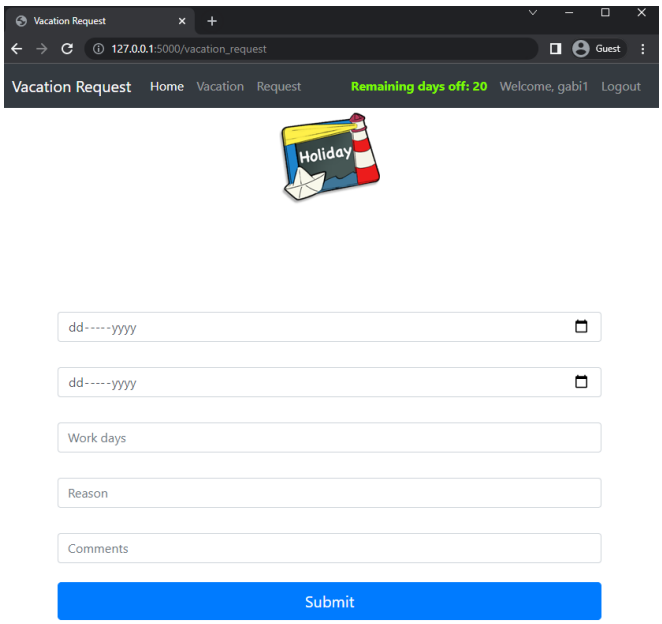

Figure 21: Vacation Request Endpoing

When the Submit button is clicked, the data is forwarded to the backend and persisted to the database.
Vacation request REST endpoint: /vacation_request

Vacations page

With the GET method of the /vacation endpoint the all user vacation requests are rendered by the browser and the current user's vacations.

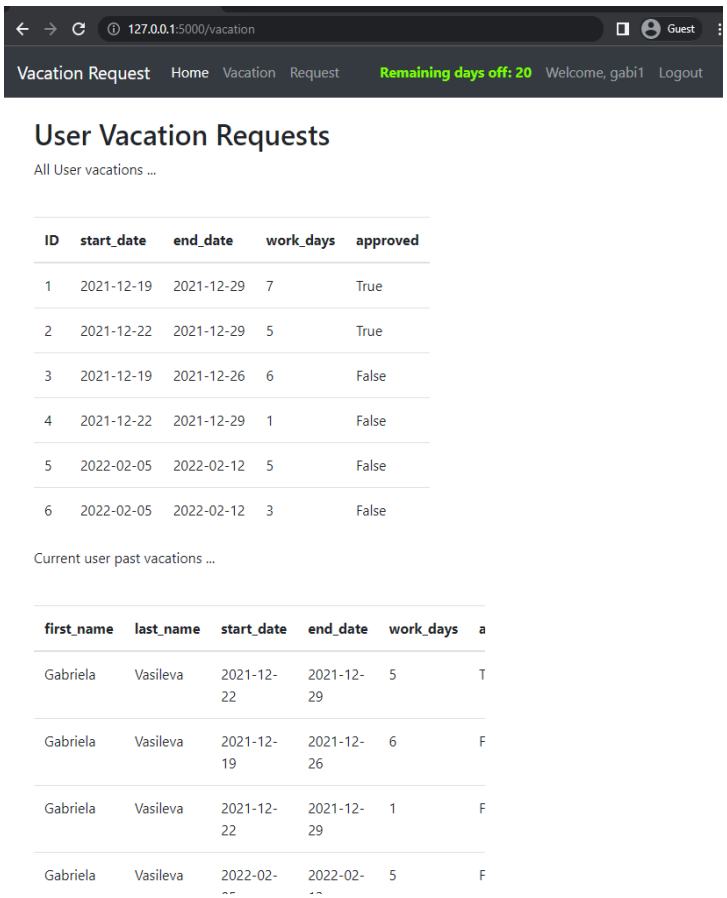

Figure 22: Vacation REST Endpoint

Vacations page REST endpoint: /vacation

Database

A database is a component in application architecture required to persist data. Data can be of many forms: structured, unstructured, semi-structured, and user state data. Structured data is the type of data that conforms to a certain structure, typically stored in a database in a normalized fashion.

Unstructured data has no definite structure. It is the heterogeneous type of data consisting of text, image files, videos, multimedia files, PDFs, blob objects, word documents, machine-generated data, etc.
Semi-structured data is a mix of structured and unstructured data. This data is often stored in data transport formats such as XML, JSON and handled as per the business requirements.
User state data is the data containing the information of activity the user performs on the website.

In our application, we are using the structured data database SQLite. SQLite is one of the most popular database technologies.

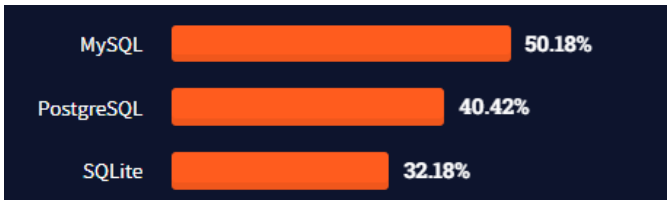

Figure 23: Top Three Database Technologies

source: https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-databases

SQLite is among the three most popular database technologies

SQLite database is implemented on a single file with no additional server process or the need for complicated RDBMS to install. All the reads and writes operations are taking place directly on a single file. SQLite comes with zero server-side software installation and minimum maintenance. In our application, we need tools that are stable and tailored to our needs. We use a relational database (RDBMS) to store our application data, so we chose to use SQLite.

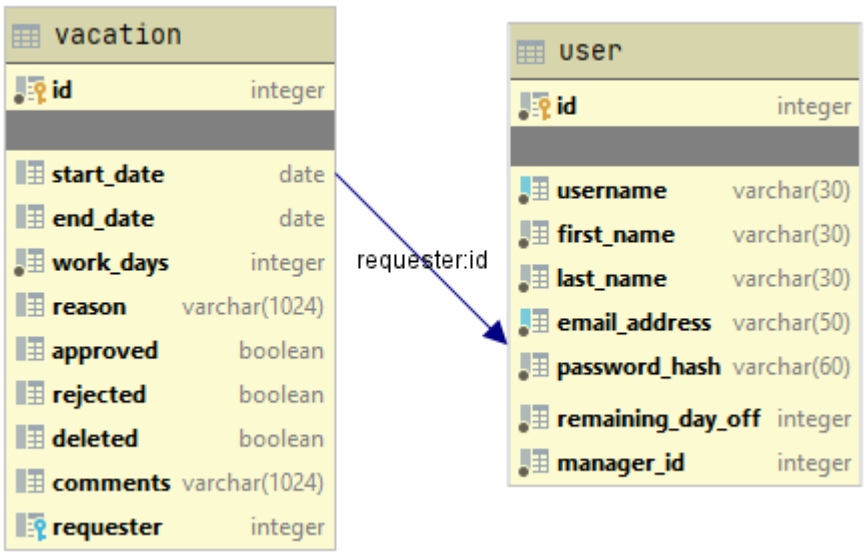

Figure 24: Application Database Relationship Architecture

The database consists of two tables vacation and user. The user table contains all the registered users of the application. Its primary columns are:

- id – of type integer which is auto-generated primary key;
- username – of type varchar limit it of 30 character which represent text, it cannot be null;
- first_name – of type varchar, it cannot be less than 2 and more than 30 chars;
- last_name – of type varchar, it cannot be less than 2 and more than 30 chars;
- email_address – of type varchar, it validated according to standard email pattern, not null;
- password_hash – of type varchar(60), it is hash of the entered password
- remaining_day_off – of type integer
- manager_id – of type integer, it references an id of another user

The vacation table contains all the registered users of the application. Its most important columns are:

- id – of type integer which is auto-generated primary key;
- requester – of type integer is a foreign key which references
- start_date – of type date
- end_date – of type date
- work_days – ot type integer,
- reason – of type varchar(1024), it can be null
- approved – of type boolean
- rejected – of type boolean
- deleted – of type boolean
- comments – of type varchar(1024), it can be null.

Framework

A framework is code storage that should help developers achieve the required result by making work easier, scalable, efficient, and maintainable web applications by providing code or extensions for common operations.

Flask gives the developer varieties of choice when developing web applications, it provides you with tools, libraries, and mechanisms that allow you to build a web application but will not enforce any dependencies or tell you how the project should look like. It requires little reliance on updates and alerts for security bugs. Flask is based on Werkzeug a WSGI utility library and Jinja2 which is its template engine. After building the application backend with Flask we connect it to the frontend best on HTML. When the user sends information on the net or goes to the search bar, the HTML connects the user. The flask framework looks for HTML files (templates) in a folder called templates. Before sending the template over, Python code is executed which injects variables, code etc.

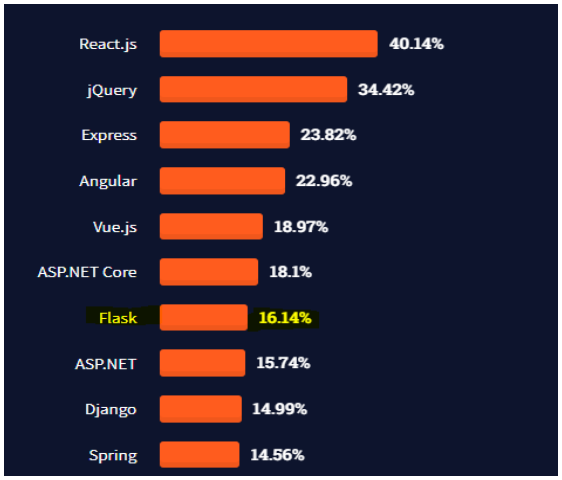

Figure 25: Top Ten Web Frameworks

source: https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks

Flask is in the top ten of the most commonly used web framework.

Development Deployment

1. Create the project virtual environment

   create:

   conda env create --prefix ./env --file environment.yml --force

   update:

   conda env update --prefix ./env --file environment.yml --prune

   To activate this environment, use

   $ conda activate [env_path]

   To deactivate an active environment, use

   $ conda deactivate

2. Setup environment variables

   set FLASK_APP=run.py

2.1. Enable debug

   set FLASK_DEBUG=1

3. Execute the application:

   flask run

Reference

1. All the diagrams are created using https://app.diagrams.net
2. Most popular programming, scripting, and markup languages https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies

Appendix

Frontend

Templates:

base.html

```
<!doctype html>
<html lang="en">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <!-- Bootstrap CSS -->
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css"
      integrity="sha384-
TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2"
crossorigin="anonymous">
  <title>
    {% block title %}

    {% endblock %}
  </title>
</head>
<body>
```

```
<nav class="navbar navbar-expand-md navbar-dark bg-dark">
  <a class="navbar-brand" href="#">Vacation Request</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarNav">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarNav">
    <ul class="navbar-nav mr-auto">
      <li class="nav-item active">
        <a class="nav-link" href="{{ url_for('home_page') }}">Home <span class="sr-
only">(current)</span></a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="{{ url_for('vacation_page') }}">Vacation</a>
      </li>

      <li class="nav-item">
        <a class="nav-link" href="{{ url_for('vacation_request_page') }}">Request</a>
      </li>
    </ul>

    {% if current_user.is_authenticated %}
    <ul class="navbar-nav">
      <li class="nav-item">
        <a class="nav-link" style="color: lawngreen; font-weight: bold ">
        Remaining days off: {{ current_user.remaining_day_off }}

        </a>
      </li>
      <li class="nav-item">
        <a class="nav-link">Welcome, {{ current_user.username }}</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="{{ url_for('logout_page') }}">Logout</a>
      </li>
    </ul>
    {% else %}
    <ul class="navbar-nav">
      <li class="nav-item">
        <a class="nav-link" href="{{ url_for('login_page') }}">Login</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="{{ url_for('register_page') }}">Register</a>
      </li>
    </ul>
    {% endif %}

  </div>
</nav>
{% with messages = get_flashed_messages(with_categories=true) %}
{% if messages %}
{% for category, message in messages %}
```

```
<div class="alert alert-{{ category }}">
  <button type="button" class="m1-2 mb-1 close" data-dismiss="alert" aria-label="Close">
     <span aria-hidden="true">&times;</span>
  </button>
  {{ message }}
</div>
{% endfor %}
{% endif %}
{% endwith %}
{% block content %}

{% endblock %}
<!-- Future Content here -->


<!-- Optional JavaScript -->
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
<script src='https://kit.fontawesome.com/a076d05399.js'></script>
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
     integrity="sha384-
DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
     crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"
     integrity="sha384-
9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6BZp6G7niu735Sk7lN"
     crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"
     integrity="sha384-
B4gt1jrGC7Jh4AgTPSdUtOBvfO8shuf57BaghqFfPlYxofvL8/KUEfYiJOMMV+rV"
     crossorigin="anonymous"></script>
</body>
{#<style>#}
{#    body {#}
{#        background-color: #212121;#}
{#        color: white#}
{#    }#}
{#</style>#}
</html>
```

home.html

```
{% extends 'base.html' %}
{% block title %}
  Welcome to vacation tracker
{% endblock %}
{% block content %}
  <div class="position-relative overflow-hidden p-3 p-md-5 m-md-3 text-center "
style="color:black">
     <div class="col-md-5 p-lg-5 mx-auto my-5">
        <h1 class="display-4 font-weight-normal">Vacation tracker</h1>
```

```
        <p class="lead font-weight-normal">VFU</p>
        <a class="btn btn-primary" href="{{ url_for('vacation_page') }}">Get Started</a>
    </div>
    <div class="product-device box-shadow d-none d-md-block"></div>
    <div class="product-device product-device-2 box-shadow d-none d-md-block"></div>
  </div>
{% endblock %}
```

login.html

```
{% extends 'base.html' %}
{% block title %}
Login Page
{% endblock %}

{% block content %}
<body class="text-center">
<div class="container">
  <form method="POST" class="form-signin" style="color:white">
    {{ form.hidden_tag() }}
    <img class="mb-4" src="https://icons.iconarchive.com/icons/custom-icon-design/flatastic-
9/96/Login-icon.png"
         alt="">
    <h1 class="h3 mb-3 font-weight-normal">
      Please Login
    </h1>
    <br>
    {{ form.username.label() }}
    {{ form.username(class="form-control", placeholder="User Name") }}

    {{ form.password.label() }}
    {{ form.password(class="form-control", placeholder="Password") }}

    <br>

    <div class="checkbox mb-3">
      <h6>Do not have an account?</h6>
      <a class="btn btn-sm btn-secondary" href="{{ url_for('register_page') }}">Register</a>
    </div>

    {{ form.submit(class="btn btn-lg btn-block btn-primary") }}

  </form>
</div>
</body>
{% endblock %}
```

register.html

```
{% extends 'base.html' %}
{% block title %}
```

```
  Register Page
{% endblock %}

{% block content %}
  <body class="text-center">
  <div class="container">
    <form method="POST" class="form-register" style="color: white">
      {{ form.hidden_tag() }}
      <img class="mb-4"
          src="https://icons.iconarchive.com/icons/custom-icon-design/flatastic-9/96/Login-
icon.png" alt="">
      <h1 class="h3 mb-3 font-weight-normal">
        Please Create your Account
      </h1>
      <br>
      {{ form.username.label() }}
      {{ form.username(class="form-control", placeholder="User Name") }}

      {{ form.first_name.label() }}
      {{ form.first_name(class="form-control", placeholder="First Name") }}

      {{ form.last_name.label() }}
      {{ form.last_name(class="form-control", placeholder="Last name") }}

      {{ form.email_address.label() }}
      {{ form.email_address(class="form-control", placeholder="Email Address") }}

      {{ form.password1.label() }}
      {{ form.password1(class="form-control", placeholder="Password") }}

      {{ form.password2.label() }}
      {{ form.password2(class="form-control", placeholder="Confirm Password") }}

      <br>

      <div class="checkbox mb-3">
        <h6>Already have an account?</h6>
        <a class="btn btn-sm btn-secondary" href="{{ url_for('login_page') }}">Login</a>
      </div>

      {{ form.submit(class="btn btn-lg btn-block btn-primary") }}
    </form>
  </div>
  </body>
{% endblock %}
```

vacation.html

```
{% extends 'base.html' %}
{% block title %}
Vacations
```

```
{% endblock %}
{% block content %}

<div class="row" style="margin-top:20px; margin-left:20px">
 <div class="col-8">
    <h2>User Vacation Requests</h2>
    <p> All User vacations ...</p>
    <br>
   <table class="table table-hover table-responsive">
     <thead>
       <tr>
         <!-- Your Columns HERE -->
         <th scope="col">ID</th>
         <th scope="col">start_date</th>
         <th scope="col">end_date</th>
         <th scope="col">work_days</th>
         <th scope="col">approved</th>
       </tr>
     </thead>
     <tbody>
       <!-- Your rows inside the table HERE: -->
       {% for vacation in all_vacations %}
       <tr>
         <td>{{ vacation.id }}</td>
         <td>{{ vacation.start_date }}</td>
         <td>{{ vacation.end_date }}</td>
          <td>{{ vacation.work_days }}</td>
         <td>{{ vacation.approved }}</td>
       </tr>
       {% endfor %}
     </tbody>
   </table>

    <p>Current user past vacations ...</p>
    <br>
    <table class="table table-hover table-responsive">
     <thead>
       <tr>
         <!-- Your Columns HERE -->
         <th scope="col">first_name</th>
          <th scope="col">last_name</th>
         <th scope="col">start_date</th>
         <th scope="col">end_date</th>
         <th scope="col">work_days</th>

         <th scope="col">approved</th>
       </tr>
     </thead>
     <tbody>
       <!-- Your rows inside the table HERE: -->
       {% for user, vacation in past_vacations %}
       <tr>
```

```
            <td>{{ user.first_name }}</td>
             <td>{{ user.last_name }}</td>
            <td>{{ vacation.start_date }}</td>
            <td>{{ vacation.end_date }}</td>
             <td>{{ vacation.work_days }}</td>

            <td>{{ vacation.approved }}</td>
          </tr>
          {% endfor %}
        </tbody>
      </table>

    </div>
  </div>
  {% endblock %}
```

vacation_request.html

```
{% extends 'base.html' %}
{% block title %}
  Vacation Request
{% endblock %}

{% block content %}
  <body class="text-center">
  <div class="container">
      <form method="POST" class="form-register" style="color: white">
        {{ form.hidden_tag() }}
        <img class="mb-4"
            src="https://icons.iconarchive.com/icons/fixicon/slate/128/holiday-icon.png" alt="">
        <h1 class="h3 mb-3 font-weight-normal">
          Please request vacation
        </h1>
        <br>
        {{ form.start_date.label() }}
        {{ form.start_date(class="form-control", placeholder="Start date") }}

        {{ form.end_date.label() }}
        {{ form.end_date(class="form-control", placeholder="End date") }}

        {{ form.work_days.label() }}
        {{ form.work_days(class="form-control", placeholder="Work days") }}

        {{ form.reason.label() }}
        {{ form.reason(class="form-control", placeholder="Reason") }}

        {{ form.comments.label() }}
        {{ form.comments(class="form-control", placeholder="Comments") }}
        <br>

        {{ form.submit(class="btn btn-lg btn-block btn-primary") }}
```

```
    </form>
  </div>
  </body>
{% endblock %}
```

Backend

__init__.py

```
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy
from flask_bcrypt import Bcrypt
from flask_login import LoginManager

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///market.db'
app.config['SECRET_KEY'] = '592809732677de08d2d9e5c1'
db = SQLAlchemy(app)
bcrypt = Bcrypt(app)
login_manager = LoginManager(app)
login_manager.login_view = "login_page"
login_manager.login_message_category = "info"
from market import routes
```

forms.py

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField,  BooleanField, TextAreaField,
PasswordField, validators, HiddenField, DateField, SelectField
from wtforms.fields import IntegerField
from wtforms.widgets import NumberInput
from wtforms.validators import Length, EqualTo, Email, DataRequired, ValidationError
from market.models import User

class RegisterForm(FlaskForm):
  def validate_username(self, username_to_check):
    user = User.query.filter_by(username=username_to_check.data).first()
    if user:
      raise ValidationError('Username already exists! Please try a different username.')

  def validate_email_address(self, email_address_to_check):
    email_address = User.query.filter_by(email_address=email_address_to_check.data).first()
    if email_address:
      raise ValidationError('Email Address already exists! Please try a different email address.')

  username = StringField(label='User Name:', validators=[Length(min=2, max=30), DataRequired()])
  first_name = StringField(label='First Name:', validators=[Length(min=2, max=30), DataRequired()])
  last_name = StringField(label='Last Name:', validators=[Length(min=2, max=30), DataRequired()])
  email_address = StringField(label='Email Address:', validators=[Email(), DataRequired()])
  password1 = PasswordField(label='Password:', validators=[Length(min=6), DataRequired()])
```

```
  password2 = PasswordField(label='Confirm Password:', validators=[EqualTo('password1'),
DataRequired()])
  submit = SubmitField(label='Create Account')

class VacationRequestForm(FlaskForm):
  start_date = DateField('Start Date:', format='%Y-%m-%d')
  end_date = DateField('End Date:', format='%Y-%m-%d')
  work_days = IntegerField("Work Days: ", widget=NumberInput(min=1, max=20, step=1))
  reason = StringField(label='Reason:')
  comments = StringField(label='Comments:')
  submit = SubmitField(label='Submit')

class LoginForm(FlaskForm):
  username = StringField(label='User Name:', validators=[DataRequired()])
  password = StringField(label='Password:', validators=[DataRequired()])
  submit = SubmitField(label='Sign in')
```

models.py

```
from market import db, login_manager
from market import bcrypt
from flask_login import UserMixin
from datetime import datetime, timedelta, date
from sqlalchemy import and_
import enum
from sqlalchemy import Integer, Enum

@login_manager.user_loader
def load_user(user_id):
  return User.query.get(int(user_id))

class User(db.Model, UserMixin):
  id = db.Column(db.Integer(), primary_key=True)
  username = db.Column(db.String(length=30), nullable=False, unique=True)
  first_name = db.Column(db.String(length=30), nullable=False)
  last_name = db.Column(db.String(length=30), nullable=False)
  email_address = db.Column(db.String(length=50), nullable=False, unique=True)
  password_hash = db.Column(db.String(length=60), nullable=False)
  budget = db.Column(db.Integer(), nullable=False, default=1000)
  remaining_day_off = db.Column(db.Integer(), nullable=False, default=20)
  manager_id = db.Column(db.Integer(), nullable=False, default=1)
  vacations = db.relationship('Vacation', backref='requester_user', lazy=True)

  @property
  def prettier_budget(self):
    if len(str(self.budget)) >= 4:
      return f'{str(self.budget)[:-3]},{str(self.budget)[-3:]}$'
    else:
      return f"{self.budget}$"

  @property
```

```
    def password(self):
        return self.password

    @password.setter
    def password(self, plain_text_password):
        self.password_hash = bcrypt.generate_password_hash(plain_text_password).decode('utf-8')

    def check_password_correction(self, attempted_password):
        return bcrypt.check_password_hash(self.password_hash, attempted_password)

    def can_purchase(self, item_obj):
        return self.budget >= item_obj.price

    def can_sell(self, item_obj):
        return item_obj in self.items

class Vacation(db.Model):
    id = db.Column(db.Integer(), primary_key=True)
    start_date = db.Column(db.Date, default = date.today())
    end_date = db.Column(db.Date, default = date.today()+ timedelta(days = 7))
    work_days = db.Column(db.Integer(), nullable=False)
    reason = db.Column(db.String(length=1024), nullable=True)
    approved = db.Column(db.Boolean(), default=False)
    rejected = db.Column(db.Boolean(), default=False)
    deleted = db.Column(db.Boolean(), default=False)
    comments = db.Column(db.String(length=1024), nullable=True)
    requester = db.Column(db.Integer(), db.ForeignKey('user.id'))
```

routes.py

```
from market import app
from flask import render_template, redirect, url_for, flash, request
from market.models import  User, Vacation
from market.forms import RegisterForm, LoginForm, VacationRequestForm
from market import db
from flask_login import login_user, logout_user, login_required, current_user

@app.route('/')
@app.route('/home')
def home_page():
    return render_template('home.html')

@app.route('/vacation', methods=['GET', 'POST'])
@login_required
def vacation_page():
    if request.method == "GET":
        all_vacations = Vacation.query.all()
        past_vacations = db.session.query(User,
Vacation).join(Vacation).filter_by(requester=current_user.id).all()
```

```
    return render_template('vacation.html', all_vacations
=all_vacations,past_vacations=past_vacations) #

@app.route('/vacation_request', methods=['GET', 'POST'])
def vacation_request_page():
  form = VacationRequestForm()

  if form.validate_on_submit():
    vacation_to_create = Vacation(
                    # start_date=form.start_date.data,
                    # end_date=form.end_date.data,
                    work_days=form.work_days.data,
                    reason=form.reason.data,
                    comments = form.comments.data,
                    requester = current_user.id )

    db.session.add(vacation_to_create)
    db.session.commit()
    flash(f"Vacation request created successfully! ", category='success')
    return redirect(url_for('vacation_page'))
  if form.errors != {}:  # If there are not errors from the validations
    for err_msg in form.errors.values():
      flash(f'There was an error with creating a vacation request: {err_msg}', category='danger')

  return render_template('vacation_request.html', form=form)

@app.route('/register', methods=['GET', 'POST'])
def register_page():
  form = RegisterForm()
  if form.validate_on_submit():
    user_to_create = User(username=form.username.data,
                    first_name=form.first_name.data,
                    last_name=form.last_name.data,
                    email_address=form.email_address.data,
                    password=form.password1.data)
    db.session.add(user_to_create)
    db.session.commit()
    login_user(user_to_create)
    flash(f"Account created successfully! You are now logged in as
{user_to_create.username}", category='success')
    return redirect(url_for('vacation_page'))
  if form.errors != {}:  # If there are not errors from the validations
    for err_msg in form.errors.values():
      flash(f'There was an error with creating a user: {err_msg}', category='danger')

  return render_template('register.html', form=form)

@app.route('/login', methods=['GET', 'POST'])
def login_page():
  form = LoginForm()
  if form.validate_on_submit():
    attempted_user = User.query.filter_by(username=form.username.data).first()
```

```
    if attempted_user and attempted_user.check_password_correction(
            attempted_password=form.password.data
    ):
        login_user(attempted_user)
        flash(f'Success! You are logged in as: {attempted_user.username}', category='success')
        return redirect(url_for('vacation_page'))
    else:
        flash('Username and password are not match! Please try again', category='danger')

  return render_template('login.html', form=form)


@app.route('/logout')
def logout_page():
  logout_user()
  flash("You have been logged out!", category='info')
  return redirect(url_for("home_page"))
```

run.py

```
from market import app

#Checks if the run.py file has executed directly and not imported
if __name__ == '__main__':
  # noinspection PyPackageRequirements
  app.run(debug=True)
```

Database

crate_database.py

```
from datetime import datetime, timedelta, date

from market import db

db.drop_all()
db.create_all()

from market.models import db

from market.models import User, Vacation

u1=User(username='gabi',first_name='Gabriela',last_name='Vasileva',manager_id = 1,
password_hash='123456', email_address='gabi@gabi.com')
u2=User(username='lily',first_name='Lili',last_name='Vasileva',manager_id = 1,
password_hash='123456', email_address='lily@lily.com')
```

```
u3=User(username='koko',first_name='Koko',last_name='Vasilev',manager_id = 1,
password_hash='123456', email_address='koko@koko.com')
u4=User(username='dan',first_name='Dan',last_name='Vasilev',manager_id = 1,
password_hash='123456', email_address='dan@dan.com')
db.session.add(u1)
db.session.add(u2)
db.session.add(u3)
db.session.add(u4)
db.session.commit()

vacation1 = Vacation(start_date = date.today(), end_date = date.today()+ timedelta(days = 10),
work_days = 7,reason = 'holiday',approved = True,comments = "fine",requester = 2)
vacation2 = Vacation(start_date = date.today()+ timedelta(days = 3), end_date = date.today()+
timedelta(days = 10), work_days = 5,reason = 'holiday',approved = True,comments =
"fine",requester = 5)

db.session.add(vacation1)
db.session.add(vacation2)
db.session.commit()
Vacation.query.all()
```

Deployment

environment.yml:

```
name: vacation-02-env

dependencies:
 - flask
 - pip:
    - flask_sqlalchemy
    - flask_bcrypt
    - Flask-Login
    - flask_wtf
    - wtforms
    - email_validator
```