به نام خدا

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

برنامه‌نویسی وب (بهار ۱۴۰۳)

پروژه پایانی

فاز بکند

Trello

ترلو (Trello)

در عصر اطلاعات، مدیریت پروژه‌ها و تیم‌ها بیش از پیش به فناوری‌های نوین وابسته شده است. ابزارهای دیجیتالی مدیریت پروژه نقش مهمی در افزایش بهره‌وری، بهبود ارتباطات تیمی، و مانیتورینگ فعالیت‌ها دارند. Trello، به عنوان یکی از پلتفرم‌های شناخته شده در این زمینه، با ارائه رابط کاربری گرافیکی بصری و کاربردی، امکان مدیریت وظایف و پروژه‌ها را به صورت سازمان‌یافته فراهم می‌کند. این پروژه با هدف توسعه یک برنامه مشابه Trello، تلاش دارد تا امکانات مشابهی را با توجه به نیازهای خاص کاربران و با امکان شخصی‌سازی بیشتر ارائه دهد.

اهداف پروژه

1. **طراحی رابط کاربری:** توسعه یک رابط کاربری گرافیکی که امکان کشیدن و رها کردن (Drag and Drop)، ویرایش، و مشاهده وضعیت پروژه‌ها و وظایف را به صورت ساده و مؤثر فراهم می‌کند.
2. **شخصی‌سازی داشبوردها:** امکان ایجاد، تنظیم، و مدیریت داشبوردهای مختلف برای نیازهای متفاوت کاربران و پروژه‌های متعدد.
3. **تعامل و همکاری:** توسعه امکاناتی برای به اشتراک گذاری داشبوردها، تعیین دسترسی‌های مختلف برای اعضای تیم، و ارتباط مؤثر بین اعضا.
4. **پشتیبانی از API:** ایجاد APIهایی برای امکان ادغام با سایر ابزارها و سیستم‌های مدیریت پروژه.
5. **امنیت و حریم خصوصی:** تأمین امنیت داده‌ها و اطلاعات کاربران با رعایت استانداردهای بین‌المللی حریم خصوصی و امنیت سایبری.

در توسعه هر نرم‌افزاری، طراحی دقیق مدل‌های داده حیاتی است. مدل‌ها ساختار داده‌هایی که برنامه بر اساس آنها عمل می‌کند را تعریف می‌کنند و به اطمینان از یکپارچگی، دسترسی بهینه و استفاده مؤثر از داده‌ها کمک می‌کنند. این مدل‌ها همچنین نقش مهمی در تعریف روابط بین داده‌های مختلف دارند، که برای حفظ سازگاری و اجرای قوانین کسب‌وکار ضروری است.

استفاده از پایگاه داده رابطه‌ای

1. **یکپارچگی داده‌ها:** مدل‌های رابطه‌ای به طور طبیعی اطمینان از یکپارچگی و دقت داده‌ها را فراهم می‌کنند، که برای پروژه‌های مدیریتی بسیار مهم است.
2. **تراکنش‌های قابل اعتماد:** سیستم‌های مدیریت پایگاه داده رابطه‌ای (RDBMS) از تراکنش‌ها پشتیبانی می‌کنند که به معنای پشتیبانی از عملیات‌های پایگاه داده به صورت اتمی، یکپارچه، مداوم و مستقل است.
3. **دسترسی بهینه:** RDBMSها دارای ابزارهای قدرتمندی برای کوئری و دسترسی به داده‌ها هستند که امکان جستجو و فیلتر کردن داده‌ها را به صورت کارآمد فراهم می‌کند.

4. **مقیاس‌پذیری و امنیت:** این سیستم‌ها معمولاً دارای ویژگی‌های پیشرفته برای مدیریت دسترسی و امنیت داده‌ها هستند، که برای برنامه‌های تجاری حیاتی است.

استفاده از پایگاه داده NoSQL

در راستای تحقق اهداف پروژه و پیشبرد فعالیت‌های مرتبط با مدیریت داده‌ها، استفاده از پایگاه‌های داده NoSQL به عنوان یک گزینه مجاز در نظر گرفته شده است. این انتخاب باید توسط تیم پروژه با ارائه دلیلی قانع کننده انجام شود تا اطمینان حاصل شود که استفاده از این تکنولوژی بهترین گزینه برای دستیابی به نیازهای خاص پروژه است.

شرایط مجاز به استفاده

- تیم پروژه باید بتواند نشان دهد که چگونه استفاده از پایگاه داده NoSQL به بهبود عملکرد، مقیاس‌پذیری و انعطاف‌پذیری پروژه کمک می‌کند.
- مستندات مربوط به تحلیل مزایا و معایب استفاده از پایگاه داده NoSQL در مقایسه با سایر گزینه‌های موجود باید به صورت کامل ارائه شود.

در صورت استفاده درست از این نوع پایگاه داده می‌تواند نمره امتیازی داشته باشد.

1. Workspace

- **Fields:**
  - `id`: Unique identifier
  - `name`: Name of the workspace
  - `description`: A brief description of the workspace
  - `created_at`: Date and time when the workspace was created
  - `updated_at`: Date and time when the workspace was last updated
- **Constraints:**
  - `name` cannot be empty and should have a valid length

2. Task

- **Fields:**
  - `id`: Unique identifier
  - `title`: Title of the task
  - `description`: Detailed description of the task
  - `status`: Current status of the task (e.g., Planned, In Progress, Completed)
  - `estimated_time`: Estimated time required to complete the task
  - `actual_time`: Actual time spent on the task
  - `due_date`: Deadline for the task
  - `priority`: Priority level of the task
  - `workspace_id`: Identifier for the workspace to which the task belongs
  - `assignee_id`: Identifier for the user to whom the task is assigned (optional)
  - `created_at`: Date and time when the task was created
  - `updated_at`: Date and time when the task was last updated
  - `image_url`: URL of the image associated with the task
- **Constraints:**
  - `title` and `workspace_id` cannot be empty.
  - `workspace_id` must be a valid reference to an existing record in the Workspace table.
  - `assignee_id` must be a valid reference to an existing record in the User table within the same workspace (optional).

3. SubTask

- **Fields:**
  - `id`: Unique identifier
  - `task_id`: Identifier of the main task

  - `title`: Title of the subtask
  - `is_completed`: Completion status of the subtask (Yes/No)
  - `assignee_id`: Identifier for the user to whom the subtask is assigned (optional)
  - `created_at`: Date and time when the subtask was created
  - `updated_at`: Date and time when the subtask was last updated
- **Constraints:**
  - `title` and `task_id` cannot be empty.
  - `task_id` must be a valid reference to an existing record in the Task table.
  - `assignee_id` must be a valid reference to an existing record in the User table within the same workspace (optional).

4. User

- **Fields:**
  - `id`: Unique identifier
  - `username`: Username of the user
  - `email`: Email address of the user
  - `password_hash`: Hashed password for user authentication
  - `created_at`: Date and time when the user account was created
  - `updated_at`: Date and time when the user account was last updated
- **Constraints:**
  - `username` and `email` must be unique and cannot be empty.
  - `email` must be valid.

5. UserWorkspaceRole (Pivot Table)

- **Fields:**
  - `id`: Unique identifier
  - `user_id`: Identifier of the user
  - `workspace_id`: Identifier of the workspace

  - `role`: Role of the user in the workspace (e.g., Admin, Standard User)
  - `created_at`: Date and time when the association was created
  - `updated_at`: Date and time when the association was last updated
- **Constraints:**
  - `user_id` and `workspace_id` cannot be empty.
  - `role` must be valid for the corresponding workspace.

تعریف اندپوینت‌های HTTP

حالا که مدل‌های پایگاه داده را طراحی کرده‌اید، نوبت این است که اندپوینت‌های HTTP را نیز طراحی و پیاده‌سازی کنید. در آغاز، توضیح داده‌ایم که برای ارتباط با سرور و مدیریت داده‌ها، باید یک سری توابع را تعریف کنیم که به ازای هر درخواست HTTP، یکی از این توابع صدا زده می‌شود. در این بخش، ما جزئیات این درخواست‌های HTTP را تعریف می‌کنیم. ما در این بخش فقط متد و مسیر درخواست HTTP را تعریف می‌کنیم و نحوه پیاده‌سازی توابع مرتبط با هر درخواست به عهده شماست. ورودی‌های سیستم نیز با توجه به پیاده‌سازی شما ممکن است متفاوت باشند، اما خروجی‌های سیستم باید به فرم JSON باشند.

برای شرح اندپوینت‌ها ما آن‌ها را دسته‌بندی می‌کنیم. هر دسته بر اساس یک مدل پایگاه داده در نظر گرفته شده است. به عنوان مثال، اندپوینت‌هایی که نیازمند اجرای عملیات بر روی مدل حساب کاربری هستند، در دسته users قرار گرفته‌اند. ما به هر یک از این دسته‌ها "resource group" می‌گوییم. برتری این روش پیاده‌سازی، قابل فهم بودن وظیفه هر اندپوینت از روی مسیر و متد درخواست آن می‌باشد. همچنین منطق پیاده‌سازی هر بخش می‌تواند مجزا از بخش‌های دیگر باشد و این ویژگی در فرآیند توسعه کار را راحت‌تر می‌کند.

حال به توضیح برخی از اندپوینت‌ها میپردازیم (توجه کنید که فقط پیاده‌سازی این اندپوینت‌ها کافی نیست و شما باید براساس نیازهای خود اندپوینت‌های دیگری را اضافه کنید.):

1. Workspace Endpoints

- GET /workspaces: Get all workspaces.
- POST /workspaces: Create a new workspace.
- GET /workspaces/{workspaceId}: Get details of a specific workspace.
- PUT /workspaces/{workspaceId}: Update details of a specific workspace.
- DELETE /workspaces/{workspaceId}: Delete a workspace.

2. Task Endpoints

- GET /workspaces/{workspaceId}/tasks: Get all tasks in a workspace.
- POST /workspaces/{workspaceId}/tasks: Create a new task in workspace.
- GET /workspaces/{workspaceId}/tasks/{taskId}: Get details of a specific task.
- PUT /workspaces/{workspaceId}/tasks/{taskId}: Update details of specific task.
- DELETE /workspaces/{workspaceId}/tasks/{taskId}: Delete a task.

3. SubTask Endpoints

- GET /tasks/{taskId}/subtasks: Get all subtasks for a specific task.
- POST /tasks/{taskId}/subtasks: Create a new subtask for a specific task.
- GET /tasks/{taskId}/subtasks/{subtaskId}: Get details of a specific subtask.
- PUT /tasks/{taskId}/subtasks/{subtaskId}: Update details of a specific subtask.
- DELETE /tasks/{taskId}/subtasks/{subtaskId}: Delete a subtask.

4. User Endpoints

- GET /users: Get all users.
- POST /users: Create a new user.
- GET /users/{userId}: Get details of a specific user.
- PUT /users/{userId}: Update details of a specific user.
- DELETE /users/{userId}: Delete a user.

5. User-Workspace Role Endpoints

- GET /workspaces/{workspaceId}/users: Get all users in a workspace.
- POST /workspaces/{workspaceId}/users: Add a user to a workspace with a role.
- PUT /workspaces/{workspaceId}/users/{userId}: Update the role of a user in a workspace.
- DELETE /workspaces/{workspaceId}/users/{userId}: Remove a user from a workspace.

6. Other Endpoints

- POST /auth/signup: Register a new user.
- POST /auth/login: Authenticate and generate a token for the user.
- GET /users/{userId}/profile: Get the profile details of a user.
- PUT /users/{userId}/profile: Update the profile details of a user.

وب سوکت چیست؟

وب‌سوکت‌ها پروتکلی برای برقراری ارتباط دوطرفه (bi-directional) و لحظه‌ای (real-time) بین یک کلاینت (مرورگر وب) و یک سرور از طریق یک اتصال واحد و پایدار هستند. ws کانال‌های ارتباطی فول‌دپلکس (full-duplex) از مرورگر وب، به سرور را بر روی یک اتصال TCP واحد، پایدار و دوطرفه فراهم می‌کند. این امر، امکان برقراری ارتباط لحظه‌ای و رویداد-محور (event-driven) بین یک کلاینت و یک سرور را می‌دهد.

وب‌سوکت‌ها بخشی از مشخصات HTML5 هستند و از سوی همه مرورگرهای وب مدرن پشتیبانی می‌شوند. آن‌ها معمولاً با استفاده از جاوااسکریپت در سمت کاربر و یک فناوری سمت سرور مانند Node.js یا جاوا در سمت سرور پیاده‌سازی می‌شوند.

برخلاف درخواست‌های سنتی HTTP که یک‌طرفه بوده و از مدل درخواست-پاسخ (request-response) پیروی می‌کند، و در نتیجه برای هر درخواست نیاز به برقراری اتصال جدیدی دارند، وب‌سوکت‌ها امکان برقراری ارتباط مداوم بین کلاینت و سرور، بدون نیاز به بررسی مداوم (polling) را فراهم می‌کنند.

وب‌سوکت چطور کار می‌کند؟

قبل از اینکه کلاینت و سرور داده را رد و بدل کنند، باید از لایه پروتکل کنترل انتقال (TCP) برای برقراری ارتباط استفاده کنند. وب‌سوکت‌ها با استفاده از پروتکل خاص خودشان، به طور موثر به عنوان یک لایه انتقال روی اتصال TCP عمل می‌کنند. وب‌سوکت‌ها با برقراری یک اتصال پایدار بین کلاینت و سرور بر روی یک سوکت TCP واحد کار می‌کنند. هنگامی که اتصال برقرار شد، داده‌ها می‌توانند به‌صورت لحظه‌ای بین کلاینت و سرور ارسال و دریافت شوند.

پروتکل وب‌سوکت از دو بخش تشکیل شده است: handshake اولیه HTTP و پروتکل خود وب‌سوکت.

handshake اولیه HTTP

handshake اولیه HTTP برای برقراری اتصال وب‌سوکت استفاده می‌شود. کلاینت یک درخواست HTTP به سرور ارسال می‌کند و پروتکل وب‌سوکت را در هدر ارتقا (Upgrade) مشخص می‌کند. درخواست handshake علاوه بر سربرگ درخواست ارتقا، شامل یک سربرگ 64 بیتی Sec-WebSocket-Key نیز می‌شود. سرور با هش کلید در یک سربرگ Sec-Websocket-Auth پاسخ می‌دهد. این تبادل سربرگ مانع از ارسال مجدد تبادل‌های قبلی وب‌سوکت توسط یک پراکسی کش می‌شود.

سرور با یک پاسخ HTTP که شامل یک هدر ارتقا است، پاسخ می‌دهد که نشان می‌دهد به پروتکل وب‌سوکت تغییر می‌کند.در واقع، پس از اتصال از طریق یک جفت درخواست/پاسخ HTTP، کلاینت‌ها می‌توانند با استفاده از یک سربرگ ارتقا در HTTP/1.1، اتصال خود را از HTTP به وب‌سوکت تغییر دهند. با این حال، بر خلاف HTTP/1.1، اتصالات وب‌سوکت کاملاً ناهمزمان هستند.

مهم است که توجه داشته باشید که هنگام اجرا روی لایه پروتکل وب‌سوکت، وب‌سوکت‌ها برای استفاده از یک طرح «ws:» یا «wss:» به یک شناسه منبع یکسان (URI) نیاز دارند، مشابه این‌که URLهای HTTP همیشه از طرح «http:» یا «https:» استفاده می‌کنند.

پروتکل وب‌سوکت

پس از تکمیل handshake اولیه HTTP، کلاینت و سرور می‌توانند با استفاده از پروتکل وب‌سوکت ارتباط برقرار کنند. پروتکل وب‌سوکت یک پروتکل ساده مبتنی بر پیام است که امکان برقراری ارتباط دوطرفه بین کلاینت و سرور را فراهم می‌کند.

پیام‌ها در فریم‌هایی ارسال می‌شوند که از یک هدر و یک (payload) تشکیل شده‌اند. هدر حاوی اطلاعاتی در مورد فریم است، مانند نوع پیام، طول و اینکه آیا آخرین فریم در یک پیام است. payload حاوی داده‌های واقعی پیام است.

در این پروژه، از آنجایی که نیاز به ارسال اعلان‌ها به کاربران و به‌روزرسانی وضعیت کاربران وجود دارد، استفاده از وب‌سوکت (WebSocket) به عنوان یک راه حل مناسب به نظر می‌رسد. وب‌سوکت امکان برقراری ارتباط دوطرفه و لحظه‌ای بین کلاینت (مرورگر وب) و سرور را از طریق یک اتصال پایدار و پایدار فراهم می‌کند.

با استفاده از وب‌سوکت، سرور می‌تواند به طور فوری اطلاعات را به کلاینت ارسال کند، مانند اعلان‌ها و به‌روزرسانی‌های وضعیت کاربران. به علاوه، کلاینت نیز می‌تواند به سرور پیام ارسال کرده و درخواست‌های خود را ارسال کند، مانند درخواست برای به‌روزرسانی وضعیت کاربر یا درخواست اطلاعات جدید.

با این رویکرد، اعلان‌ها به‌روزرسانی شده و وضعیت کاربران به صورت لحظه‌ای و به‌موقع نمایش داده می‌شود، که تجربه کاربری بهتری را ارائه می‌دهد. در نتیجه، استفاده از وب‌سوکت به عنوان یک راهکار موثر و مناسب برای پروژه شما توصیه می‌شود.

احراز هویت (Authentication)

هنگام استفاده از APIهای حساس و مشخص کردن اینکه کدام کاربر APIها را صدا می‌زند، نیاز داریم تا فرد مورد نظر با دادن اطلاعات شخصی و مناسب خودش هویتش را تایید کند و موقع صدا زدن APIها هر بار مشخص باشد که خودش آن را صدا کرده! برای Authentication در APIها، رایج‌ترین روش استفاده از یک توکن در سربار (Header) درخواست HTTP است.

روش‌های متفاوتی برای پیاده‌سازی Auth وجود دارد. برخی از روش‌های رایج این روش عبارتند از:

- **نام کاربری و رمز عبور:** این روش از یک نام کاربری منحصر به فرد و یک رمز عبور برای تأیید هویت کاربر استفاده می‌کند.
- **تأیید دو مرحله‌ای (MFA):** این روش به رمز عبور یک لایه امنیتی دیگر مانند کد موقت ارسال شده به تلفن همراه کاربر اضافه می‌کند.

- **تأیید مبتنی بر توکن:** در این روش، کاربر یک توکن منحصر به فرد دریافت می‌کند که برای مدت زمان محدود معتبر است.

احراز هویت (Authentication) و مجوزدهی (Authorization) در API احراز هویت و مجوزدهی در API فرآیندی برای مدیریت کاربران است که هویت کاربر یا برنامه‌ی کاربردی‌ای که درخواست دسترسی به یک API را می‌کند، تأیید می‌کند.

- **احراز هویت API:** فرآیندی برای تأیید هویت کاربر یا برنامه‌ای است که درخواست را ارسال می‌کند.
- **مجوزدهی API:** فرآیندی برای تأیید این است که کاربر یا برنامه‌ی تأیید شده، اجازه‌ی دسترسی به منابع درخواستی را دارد.

احراز هویت API می‌تواند با روش‌های مختلفی انجام شود، مانند:

- **ارائه نام کاربری و رمز عبور:** روشی سنتی که در آن کاربر یک نام کاربری منحصر به فرد و یک رمز عبور برای تأیید هویت ارائه می‌دهد.
- **استفاده از سیستم مبتنی بر توکن:** مانند OAuth یا JWT. در این روش، کاربر یک توکن منحصر به فرد دریافت می‌کند.

مجوزدهی API معمولاً با استفاده از Access Token انجام می‌شود. این توکن‌ها پس از Authentication موفق به کاربر داده می‌شوند و برای دسترسی به منابع خاص برای مدت زمان محدود قابل استفاده هستند.

هدف از احراز هویت و مجوزدهی API اطمینان از این است که فقط کاربران یا برنامه‌های مجاز بتوانند به API و منابعی که ارائه می‌دهد دسترسی داشته باشند. این کار به محافظت از داده‌های حساس کمک می‌کند و اطمینان می‌دهد که API به روشی سازگار با هدف مورد نظر استفاده می‌شود.

روش‌های تایید هویت اولیه (Basic Authentication) و تایید هویت با کلید و تایید هویت مبتنی بر JWT توضیح داده شدند.

کاربرد JWT در تایید هویت:

1. **ایجاد JWT پس از ورود:**
   - زمانی که کاربر وارد برنامه وب می‌شود، سرور یک JWT ایجاد می‌کند. (Login API)
   - این JWT حاوی اطلاعاتی در مورد کاربر، مانند شناسه‌ی کاربری و مجوزها است. (درواقع بسته به پروژه مقدارهای داده‌های JWT می‌تواند متفاوت باشد.)
   - JWT با استفاده از یک کلید مخفی (private key) که فقط سرور از آن اطلاع دارد، امضا می‌شود.
2. **ارسال JWT به کاربر:**
   - سرور JWT را به کاربر ارسال می‌کند.
3. **استفاده از JWT در درخواست‌های بعدی:**
   - کاربر می‌تواند از JWT برای دسترسی به منابع حفاظت‌شده روی سرور (APIها) استفاده کند.
   - به این منظور، JWT را در سربار Authorization درخواست‌های بعدی قرار می‌دهد.
4. **اعتبارسنجی JWT در سرور:**
   - سرور امضای JWT را با استفاده از کلید مخفی خود بررسی می‌کند.
   - اگر امضا معتبر باشد، سرور داده موجود در JWT را استخراج می‌کند.

- سرور از این داده‌ها را برای تایید دسترسی کاربر به منابع درخواستی استفاده می‌کند.

مزایای JWT:

- **امنیت:** به دلیل استفاده از امضا و کلید مخفی، جعل یا دستکاری JWT دشوار است.
- **بدون نیاز به سِرور مرکزی (Stateless):** سرور نیازی به ذخیره‌ی وضعیت ورود کاربران ندارد و این امر مقیاس‌پذیری را افزایش می‌دهد.
- **قابل حمل (Portable) :** را می‌توان بین سرویس‌ها و سیستم‌های مختلف به اشتراک گذاشت.

امنیت

امنیت در توسعه‌ی وب از اهمیت بالایی برخوردار است، زیرا از داده‌های حساس کاربران و اطلاعات برنامه در برابر حملات و سوءاستفاده‌ها محافظت می‌کند. در اینجا به برخی از معیارهای کلیدی امنیتی که باید در پروژه‌های بک‌اند در نظر گرفته شوند، اشاره می‌کنیم:

1. احراز هویت و مجوزدهی:

- **احراز هویت قوی:** از روش‌های احراز هویت قوی مانند نام کاربری و رمز عبور منحصر به فرد، احراز هویت دو مرحله‌ای (2FA) یا احراز هویت مبتنی بر توکن استفاده کنید.

  پیاده سازی: در پروژه می‌توانید شرط اینکه رمزعبور کاربر بیش از ۸ کلمه باشد، دارای حداقل یک حرف کوچک، یک حرف بزرگ، یک عدد و یک علامت باشد قرار دهید!

- **مجوزدهی دقیق:** دسترسی کاربران به منابع و قابلیت‌ها را به طور دقیق کنترل کنید. از مدل‌های مجوزدهی مانند RBAC (کنترل دسترسی مبتنی بر نقش) یا ABAC (کنترل دسترسی مبتنی بر قابلیت) استفاده کنید.

2. حفاظت از داده‌ها:

- **هش کردن رمز عبور:** رمز عبور کاربران را با استفاده از الگوریتم‌های رمزنگاری قوی مانند SHA256 ذخیره کنید.

پیاده سازی: هنگام ذخیره سازی رمز کاربر پس از ثبت‌نام، به جای متن، هش آن را در دیتابیس ذخیره کنید. هنگام اعتباری سنجی رمز ورودی، کافیست هش رمز ورودی را با داده دیتابیس مطابقت دهید.

- **رمزنگاری داده‌ها:** داده‌های حساس را در هنگام ذخیره‌سازی و انتقال، به طور ایمن رمزنگاری کنید. از الگوریتم‌های رمزنگاری قوی مانند AES یا RSA استفاده کنید.
- **کنترل دسترسی به پایگاه داده:** دسترسی به پایگاه داده را به طور دقیق کنترل کنید و فقط به کاربران و برنامه‌های مجاز اجازه دسترسی به آن را بدهید. درواقع از role management استفاده کنید. برای مثال برای وصل شدن به دیتابیس توسط پروژه، نباید با یوزر اصلی وصل شد، و باید یک کاربر جدید ساخت که طبق پروژه دسترسی داشته باشد. بهترین حالت پیاده سازی این مسئله دادن دسترسی Read, Update و فقط پیاده سازی این دو مورد در دیتابیس است! یعنی هیچوقت داده را به طور مستقیم حذف نکنیم.

3. محافظت از برنامه:

- **به‌روزرسانی‌های امنیتی:** به طور مرتب سیستم‌عامل، کتابخانه‌ها و چارچوب‌های نرم‌افزاری خود را به روز کنید تا از وصله‌های امنیتی جدید بهره‌مند شوید.

5. استفاده از ابزارهای امنیتی:

- از فایروال‌ها، سیستم‌های تشخیص نفوذ (IDS) و سیستم‌های پیشگیری از نفوذ (IPS) برای محافظت از برنامه خود در برابر حملات خارجی استفاده کنید.
- از ابزارهای امنیتی برنامه‌نویسی مانند اسکنرهای کد استاتیک و کتابخانه‌های امنیتی برای محافظت از برنامه خود در برابر آسیب‌پذیری‌های داخلی استفاده کنید.

نکات تکمیلی:

- از رمزگذاری SSL/TLS برای محافظت از ترافیک شبکه خود استفاده کنید.
- از سیاست‌های قوی برای مدیریت گذرواژه‌ها و رمزهای API استفاده کنید.
- سوابق امنیتی را به طور کامل ثبت و نگهداری کنید.

پیاده‌سازی Authorization و Authentication

پیاده‌سازی ساده از Authentication می‌تواند به ترتیب زیر باشد.

1. یک API Login صدا زده می‌شود و کاربر اطلاعات ورود را با API post وارد می‌کند. (Username Password, Sign in with google, etc...)
2. سپس پروژه Access Token و Refresh Token مخصوص کاربر می‌سازد. (در صورت وجود) برای امنیت بالا Access Token طول عمر ۱۵ الی ۳۰ دقیقه و Refresh Token یک روز پیاده می‌شود. (البته بسته به حساس بودن پروژه این مقادیر می‌توانند متفاوت باشند!)
3. کاربر (پروژه Front-End) موقع صدا زدن API در سربار توکن Access را قرار داده و این توسط پروژه چک می‌شود.
4. هنگامی که دیگر توکن معتبر نباشد API Refresh توسط کاربر (کلاینت) صدا زده شده که بدنه داده درخواست آن Refresh Token و هدر آن Access Token است. پروژه بلافاصله در خروجی Access Token جدید مخصوص رفرش توکن ورودی می‌سازد.

روند فوق در ارتباط بین Client و Server انجام می‌شود.

حال برای Authorization بررسی می‌کنیم:

از آنجایی که در پروژه نیاز است تا دسترسی‌های ساخت Workspace، اضافه کردن عضو جدید به یک Workspace و حذف آن توسط ADMIN انجام شود، باید یک سطح کاربری Admin تعریف شود و برای هر workspace تعریف شود.

دیدن یک Workspace، اضافه کردن تسک و ادیت تسک و هر کار مربوط به آن، فقط توسط کاربران عضو آن Workspace قابل انجام است!

درواقع در Authorization هر کاربر فقط می‌تواند بخش‌هایی را که دسترسی دارد ببیند و نباید دسترسی بیش از حد داشته باشد!

Docker (امتیازی)

داکر یک پلتفرم متن‌باز است که برای توسعه، تست، و اجرای برنامه‌ها با استفاده از کانتینرها طراحی شده است. کانتینرها به شما امکان می‌دهند تا برنامه‌های خود را در محیطی ایزوله اجرا کنید، که این امر موجب می‌شود برنامه‌ها در محیط‌های مختلف به طور یکسان کار کنند و به راحتی قابل حمل باشند. داکر مانند یک ماشین مجازی عمل می‌کند، ولی سبک‌تر است و منابع کمتری نسبت به ماشین‌های مجازی سنتی استفاده می‌کند. این تکنولوژی مبتنی بر فناوری کانتینرهای لینوکس است و یکی از محبوب‌ترین ابزارها در دنیای DevOps است.

برای ساخت یک image داکر، اولین قدم نوشتن یک Dockerfile است، Dockerfile یک فایل است که دستورالعمل‌هایی را برای ساخت image شامل می‌شود. در این فایل، شما مشخص می‌کنید که داکر چگونه محیط را برای اجرای برنامه شما آماده کند.

Docker Compose (امتیازی)

docker-compose ابزاری است که اجرای چندین کانتینر داکر را در یک پروژه تسهیل می‌بخشد. برای استفاده از آن، شما باید یک فایل docker-compose.yaml بنویسید که تنظیمات مورد نیاز برای اجرای سرویس‌های مختلف را مشخص می‌کند.

- برای پروژه خود یک dockerfile بنویسید، سپس فایل docker-compose را برای دیپلوی کردن پروژه خود و تمام سرویس‌های مورد نیاز آن بنویسید. کانفیگ‌های وب‌سرور باید به صورت متغیرهای محیطی[1] در این فایل قابل تغییر باشند.

مانیتورینگ (امتیازی)

در دنیای اینترنت، monitoring یا نظارت بر سیستم‌ها، نقش حیاتی در اطمینان از عملکرد، پایداری و امنیت سیستم‌های کامپیوتری ایفا می‌کند. ابزارهای monitoring به مدیران سیستم اجازه می‌دهند تا دید مفیدی از وضعیت زیرساخت‌های خود کسب کنند، از وضعیت سلامت سخت‌افزار و نرم‌افزار گرفته تا تحلیل عملکرد و کشف نقاط ضعف احتمالی، پیش از آنکه به مشکلات جدی تبدیل شوند.
ابزارهایی مانند Prometheus و Grafana برای مدیریت و نمایش داده‌های نظارتی در سیستم‌های مختلفی، از جمله وب‌سرورها به کار می‌روند، که به تیم‌های فنی کمک می‌کنند تصمیمات آگاهانه‌تری بر اساس داده‌های واقعی بگیرند.

[1] Environment Variable

Prometheus

یک سیستم نظارتی و هشدار متن‌باز است که برای ثبت وقایع و متریک‌های زمان‌دار در محیط‌های سیستم‌های کامپیوتری توسعه یافته است. Prometheus به طور گسترده برای نظارت بر برنامه‌های کاربردی و زیرساخت‌ها استفاده می‌شود.
Prometheus با استفاده از مدل سرکشی[2] عمل می‌کند؛ به این معنا که Prometheus به طور دوره‌ای درخواست‌های HTTP را به سرویس‌هایی که نظارت می‌کند ارسال داشته و داده‌های متریک را از آنها دریافت می‌کند. هر سرویس باید یک endpoint موسوم به **"metrics/"** داشته باشد که Prometheus می‌تواند از آن داده‌های متریک را دریافت کند.
متریک‌ها به صورت برچسب‌دار[3] ذخیره می‌شوند که این امر تجزیه و تحلیل و فیلتر کردن داده‌ها را بر اساس معیارهای مختلف امکان‌پذیر می‌سازد. Prometheus همچنین دارای قابلیت زبان پرس و جوی[4] خود به نام PromQL است که برای استخراج و تحلیل داده‌ها به کار می‌رود.

Grafana

یک پلتفرم تجزیه و تحلیل و نظارت متن‌باز است که به کاربران امکان می‌دهد داشبوردهای دینامیک و بصری از داده‌های متنوع ایجاد کنند. Grafana به طور ویژه برای کار با داده‌های زمانی مانند آنچه توسط Prometheus جمع‌آوری می‌شود، طراحی شده است.
پس از اتصال Prometheus به عنوان یک منبع داده[5] در Grafana، می‌توانید داشبوردهایی را برای نمایش داده‌های متنوع ایجاد و سفارشی کنید. در Grafana، از پنل‌ها برای نمایش انواع متریک‌ها استفاده می‌شود، مانند نمودارهای خطی، نمودارهای میله‌ای، و جداول.، شما می‌توانید پنل‌های مختلفی را تنظیم کرده و به هر پنل یک یا چند کوئری از Prometheus اختصاص دهید. برای مثال، یک پرس‌وجو می‌تواند تعداد درخواست‌های دریافتی توسط یک سرور را نشان دهد یا متوسط زمان پاسخ‌دهی را محاسبه کند. این داشبوردها به صورت زنده بروزرسانی می‌شوند و امکان مشاهده سریع تغییرات در داده‌ها را در زمان واقعی فراهم می‌آورند.

در پرومتئوس، متریک‌های زیر را جمع‌اوری کنید:

- تعداد کل درخواست‌ها، به تفکیک درخواست‌های موفق و ناموفق
- تعداد کل درخواست‌های سرویس به پایگاه‌داده، به تفکیک درخواست‌های موفق و ناموفق
- تاخیر کل درخواست‌های مختلف

سپس با استفاده از این داده‌ها، یک داشبورد گرافانا برای نمایش موارد بالا ایجاد کنید.

---

[2] Polling
[3] Labeled
[4] Querying
[5] Data Source