مهندسی اینترنت **تمرین شماره‌ی ۷**

مدرس: **دکتر خامس‌پناه** طراحان: رضا قوی، محمد فرهی

مهلت تحویل: سه شنبه ۹ خرداد ۱۴۰۱، ساعت ۲۳:۵۵

Baloot

مقدمه

هدف از این پروژه، آشنایی با روش‌های احراز هویت[1] و کسب اجازه[2] و اطمینان از وجود برخی از پارامترهای امنیتی در برنامه می‌باشد.

[1] Authentication

[2] Authorization

فاز هفتم پروژه

فرآیند ثبت‌نام کاربران

در این قسمت قرار است فرآیند ثبت‌نام کاربران را طراحی کنید که در آن پس از وارد کردن فیلدهای لازم، یک کاربر جدید با مشخصات داده شده به پایگاه‌داده اضافه شود. در این صفحه فیلدهای **نام کاربری، تاریخ تولد، آدرس، ایمیل** و **کلمه‌ی عبور و ...** از کاربر دریافت می‌شود.

نکات:

- قبل از ارسال اطلاعات به سرور، فرمت ورودی ها را در سمت کاربر[3] اعتبارسنجی کنید.
- کلمه‌ی عبور را به هیچ وجه به صورت plain text در پایگاه‌داده ذخیره نکنید بلکه از hash آن استفاده کنید.
- انجام اعتبارسنجی‌هایی نظیر تکراری نبودن ایمیل در سمت سرور الزامیست. اگر ایمیل تکراری باشد، باید خطای مناسب به کاربر نمایش داده شود.

مانند فازهای قبلی، همچنان دریافت لیست کاربران در هنگام اجرا شدن پروژه و اضافه کردن آن‌ها به پایگاه‌داده را خواهید داشت و فرآیند ثبت‌نام، صرفا برای کاربران جدید است. در این تمرین، کلمه‌ی عبور تمام کاربران را به صورت hash شده ذخیره کنید.

http://5.253.25.110:5000/api/users

[3] Client

احراز هویت به کمک JWT[4]

در این بخش به کمک JWT که یک روش بدون حالت[5] است (بدون نیاز به حافظه در سمت سرور)، احراز هویت را به برنامه‌ی خود اضافه می‌کنید. استاندارد JWT در اکثر زبان‌های برنامه‌سازی پیاده‌سازی شده و برای جاوا نیز چندین پیاده‌سازی برای آن وجود دارد.

هر JWT شامل سه بخش است:

۱. Header: شامل اطلاعات الگوریتم مورد استفاده برای signature و نوع token است.

۲. Payload: شامل claim های JWT است. در این پروژه استفاده از claim های **iss**، **iat** و **exp** (با زمان انقضای یک روز) اجباری است. در کنار آن‌ها می‌توانید از claim های استاندارد یا غیراستاندارد دیگر نیز استفاده کنید (مثلاً یک claim به نام userEmail برای هویت کاربر).

۳. Signature: این قسمت شامل امضای دیجیتال سرور است که برای اطمینان از صحت JWT اضافه می‌شود. این امضا معمولا به کمک الگوریتم‌های HMAC و RSA محاسبه می‌شود. در این تمرین برای راحتی از **الگوریتم HMACSHA256** همراه با **کلید baloot2023** استفاده کنید. در این حالت امضا به‌صورت زیر تولید می‌شود:

HMACSHA256(base64UrlEncode(header) + '.' + base64UrlEncode(payload), "baloot2023")

به دلیل اینکه signature تنها توسط سرور قابل تولید است (چون فقط سرور کلید را دارد)، پس میتوان صحت JWT را بر اساس آن سنجید. در صورت علاقه می‌توانید کمی در مورد امضای دیجیتال تحقیق کنید.

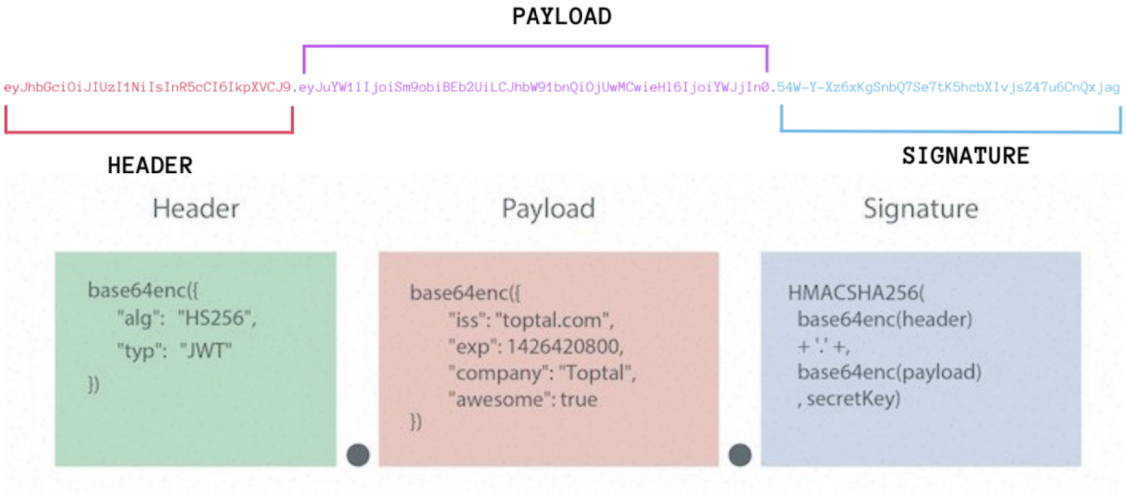
---

[4] JSON Web Token

[5] Stateless

فرمت نهایی JWT به‌صورت زیر است:

base64UrlEncode(header) + '.' + base64UrlEncode(payload) + '.' + signature



فرایند ورود کاربر

در این قسمت قرار است فرایند ورود کاربر را پیاده‌سازی کنید. ابتدا لازم است یک API طراحی کنید که ایمیل و کلمه‌ی عبور کاربر را دریافت کند و در صورت درست بودن اطلاعات، برای کاربر یک JWT صادر کند. در صورت عدم صحت اطلاعات نیز پیغام مناسب همراه با status code شماره‌ی 401 به کاربر می‌فرستد.

در سمت کاربر نیز پس از صادر شدن JWT، این توکن را نگه‌داری کنید و این توکن را در درخواستهای بعدی در Authorization Header بفرستید.

احراز هویت با مکانیزم OAuth

در این بخش، شما باید امکان ورود کاربر با استفاده از سرویس GitHub را پیاده سازی نمایید.

برای این کار لازم است ابتدا یک OAuth Application در Github ایجاد کنید. نحوه ایجاد اپلیکیشن در این لینک موجود است.

سپس لینکی را که کاربر باید بر روی آن کلیک کند تا به GitHub ریدایرکت شود و به اپلیکیشن شما دسترسی بدهد، در صفحه لاگین و ثبت‌نام قرار دهید. این لینک به شکل زیر خواهد بود:

https://github.com/login/oauth/authorize?client_id=CLIENT_ID&scope=user

که باید به جای CLIENT_ID از مقداری که هنگام ایجاد اپلیکیشن در GitHub تولید شده‌است، استفاده کنید.

سپس لازم است یک صفحه Callback در بخش فرانت‌اند و یک اندپوینت Callback در بخش بک‌اند پروژه خود پیاده سازی نمایید.

صفحه Callback فرانت‌اند، یک صفحه خالی است که کاربر پس از دادن دسترسی به اپلیکیشن شما، با پارامتر "code" به آن ریدایرکت می‌شود. این کد توسط GitHub ایجاد شده و با ریدایرکت کردن کاربر به صفحه Callback، در واقع آن را در اختیار اپلیکیشن شما قرار می‌دهد.

صفحه Callback لازم است بلافاصله پس از load شدن، اندپوینت Callback بک‌اند را با همان کد داده شده در پارامتر، فراخوانی کند و در پاسخ، توکن JWT مربوط به کاربر را از بک‌اند دریافت کرده و کاربر را به صفحه خانه هدایت کند.

در بدنه Callback پیاده سازی شده در بک‌اند، لازم است ابتدا یک درخواست به GitHub برای دریافت Access Token کاربر ارسال شود. این درخواست باید شامل code داده شده از بخش فرانت‌اند، client_id و client_secret باشد.

این درخواست باید با متد POST و به آدرس زیر ارسال شود (توجه کنید برای آن که پاسخ به صورت json برگردانده شود، هدر Accept را برابر application/json قرار دهید):

https://github.com/login/oauth/access_token?client_id=CLIENT_ID&client_secret=CLIENT_SECRET&code=CODE

client_id و client_secret هنگام ایجاد اپلیکیشن در GitHub به شما داده می‌شود. با وجود این 2 پارامتر در درخواست ارسالی، GitHub اپلیکیشن شما را احراز هویت کرده و با توجه به code داده شده، اجازه دریافت Access Token کاربر را به شما می‌دهد و آن را در پاسخ با کلید access_token، برمی‌گرداند. از این پس با قرار دادن این توکن در درخواست‌های خود به GitHub API، می‌توانید اطلاعاتی را که کاربر اجازه دسترسی آن‌ها را به شما داده است، دریافت کنید.

برای این پروژه فقط لازم است اطلاعات کاربر را از اندپوینت زیر دریافت کنید:

https://api.github.com/user

از این لینک می‌توانید سند مربوط به این API را مشاهده کنید.

در درخواست به API باید Access Token دریافتی را در هدر Authorization به صورت زیر قرار دهید:

Authorization: “token ACCESS_TOKEN”

در میان اطلاعات کاربر، برای ما تنها ایمیل (email)، نام کاربری (login) و نام کاربر (name) (همون username فرض شود.) اهمیت دارد.

پس از دریافت اطلاعات کاربر، یک کاربر جدید در دیتابیس خود ایجاد کنید یا اگر ایمیل مورد نظر موجود بود، اطلاعات کاربر را ***به‌روز*** کنید. سپس برای کاربر مورد نظر یک توکن JWT ایجاد کرده و به عنوان خروجی برگردانید تا فرایند احراز هویت کاربر تمام شود.

برای راهنمایی بیشتر درمورد GitHub OAuth Authentication می‌توانید این لینک را مطالعه نمایید.

نکات:

- GitHub تاریخ تولد کاربر را نگهداری نمی‌کند. بنابراین برای تاریخ تولد، مقدار created_at را که GitHub همراه با اطلاعات کاربر برمی‌گرداند، منهای 18 سال کرده و به عنوان تاریخ تولد ذخیره کنید.
- اگر ایمیل کاربر در گیت‌هاب public نباشد، در خروجی برگردانده شده مقدار ایمیل null خواهد بود. بنابراین برای راحتی کار، فرض کنید کاربر حتما ایمیل public برای خود قرار داده است.
- برای کاربران **جدید** که با این روش احراز هویت می‌شوند، رمز عبور (و تمام فیلد هایی که مقداری ندارد) را در دیتابیس null (یا یک مقدار پیش فرض)در نظر بگیرید.

اعتبارسنجی کاربر

پس از موفقیت آمیز بودن فرآیند ورود، یک JWT برای کاربر صادر می‌شود که کاربر برای درخواست‌های بعدی خود این JWT را به سرور می‌فرستد. در نتیجه باید در درخواست‌های بعدی بر اساس JWT فرستاده شده، اعتبارسنجی کاربر را انجام دهید.

برای این کار به‌جای اینکه در ابتدای هر servlet به بررسی این موضوع بپردازید، از فیلتر که یکی از پرکاربردترین امکانات JavaEE است، استفاده کنید. یک فیلتر بسازید و در آن درستی JWT دریافت شده را بررسی‌کنید. در پیاده‌سازی این فیلتر باید سه حالت زیر را در نظر بگیرید:

- در صورت درست بودن JWT، اطلاعات کاربر را از پایگاه‌داده بگیرید و به عنوان یک attribute برای request تعیین کنید تا در ادامه به راحتی به این اطلاعات دسترسی داشته باشید.

- در صورت وجود مشکل در JWT، پاسخی با status code شماره‌ی 401 را برای کاربر ارسال کنید.
- در صورتی‌که کاربر JWT را ارسال نکرده بود و قصد دسترسی به صفحاتی را که نیازمند احراز هویت کاربر هستند داشت، پاسخی با status code شماره‌ی 401 به او ارسال کنید.

پس از پیاده سازی این فیلتر، آن را بر روی همه‌ی API های موجود در سیستم که نیاز به احراز هویت دارند، بگذارید. دقت کنید که API های ثبت‌نام و ورود و Callback نیازی به احراز هویت ندارند.

نیازمندی های سمت رابط کاربری

در سمت کاربر دو حالت برای احراز هویت وجود دارد:

- کاربر وارد سیستم نشده و JWT ندارد که در این حالت **تنها** می‌تواند صفحات ورود و ثبت‌نام را مشاهده‌کند (در صورت واردکردن آدرس سایر صفحات، کاربر را به صفحه‌ی ورود هدایت کنید).
- کاربر به سیستم وارد شده که در این حالت می‌تواند به تمامی صفحات به جز ورود و ثبت‌نام دسترسی پیدا کند (در صورت واردکردن آدرس این صفحات، کاربر را به صفحه‌ی اصلی هدایت کنید).

در صورتی که کاربر وارد برنامه شده باشد، با بازنشانی[6] صفحه همچنان باید اطلاعات مربوط به احراز هویت او ثابت باقی بماند. برای این کار می‌توانید JWT را در حافظه‌ی محلی مرورگر نگه‌داری کنید و در هر بار بازخوانی صفحه آن را از حافظه‌ی محلی مرورگر بخوانید.

امکان خروج[7] کاربر از حسابش را نیز اضافه کنید که با توجه به بدون حالت بودن JWT، تنها کافی است این توکن را در سمت‌کاربر پاک کنید.

هدف اصلی این پروژه یادگیری فرآیندهای مربوط به احراز هویت کاربران است. به همین دلیل مسائل مربوط به طراحی و زیبایی صفحات اهمیتی ندارد و می‌توانید طراحی دلخواه خود را داشته باشید.

ایمن سازی در برابر حملات SQL Injection

در حملات Injection فرد حمله کننده در داده‌های ارسالی خود از دستورات یا کوئری‌هایی استفاده می‌کند که در صورت اجرا شدن در سرور، می‌توانند مشکل‌زا باشد. حمله‌ی SQL Injection نیز نوعی از حمله‌ی Injection است که در آن فرد حمله‌کننده کوئری‌های SQL را در داده‌های ارسالی خود به سرور می‌فرستد. به عنوان مثال می‌توانید به سناریوی عکس (۱) دقت کنید که در آن فرد حمله کننده با استفاده از حمله‌ی SQL Injection توانسته به اطلاعات تمامی دانش‌آموزان دسترسی پیدا کند.

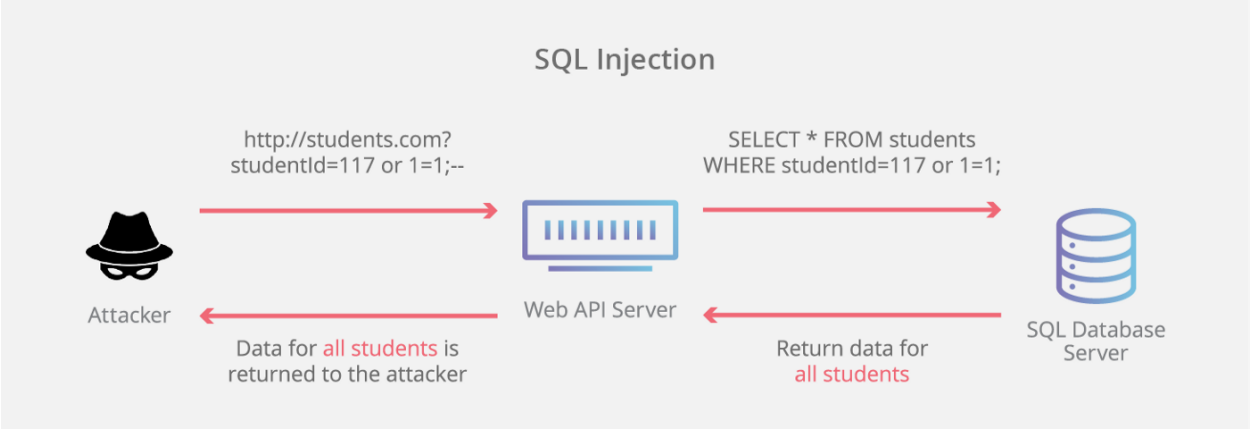
در این بخش شما باید API هایی را که در آن از کاربران ورودی دریافت می‌کنید، طوری تغییر دهید که در برابر حملات SQL Injection مقاوم باشند. برای این کار باید از

---

[6] Refresh

[7] Logout

PreparedStatement استفاده کنید. روش دیگر، پیاده‌سازی نیازمندی‌های مربوط به ذخیره‌سازی داده با کتابخانه Hibernate است.



عکس (۱)

نکات پایانی

- کافی است که یکی از اعضای گروه Hash مربوط به آخرین کامیت پروژه سمت سرور و سمت کاربر را در سایت درس آپلود کند. در هنگام تحویل، پروژه روی این کامیت مورد ارزیابی قرار می‌گیرد.
- ساختار صحیح و تمیزی کد برنامه، بخشی از نمره‌ی این فاز پروژه‌ی شما خواهد بود. بنابراین در طراحی ساختار برنامه دقت به خرج دهید.
- هدف این تمرین یادگیری شماست. لطفاً تمرین را خودتان انجام دهید. در صورت مشاهده‌ی مشابهت بین کدهای دو گروه، از نمره هر دو گروه مطابق سیاستی که در کلاس گفته شده است کسر خواهد شد.
- سوالات خود را تا حد ممکن در فروم درس مطرح کنید تا سایر دانشجویان نیز از پاسخ آن‌ها بهره‌مند شوند. در صورتی که قصد مطرح کردن سوال خاص‌تری داشتید، از طریق ایمیل با طراحان این فاز پروژه ارتباط برقرار کنید.
- ایمیل طراحان پروژه: rezaqavi1379@gmail.com و farahim.1379@gmail.com