
Nwen304 Group Project

Workload Distribution

Taichi

- Google OAuth
- Ensuring Authentication
- Dynamic Endpoints
- Use Session and Cookies
- Privacy - Hashing
- Password Reset

James

- Hosted (Cloud Run)
- MongoDB
- environments
- Structure MVC architecture
- Microservices

Together

- Schema
- Basic Login/Logout Function
- Recommendation Services
- Testing
- Timeout Function

Bobo

Model

Schemas for Database

User Schema:

```
const mongoose = require('mongoose');
const UserSchema = new mongoose.Schema({
  username: String,
  email: String,
  password: String,
  resetToken: String,
  resetTokenExpiration: Date,
  location: String, //for recommendation
  purchaseHistory: [String]
});
const User = mongoose.model('User', UserSchema);
module.exports = User;
```

Item Schema:

```
const mongoose = require('mongoose');
const itemSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true
  },
  description: {
    type: String,
  },
  price: {
    type: Number,
  },
  image: {
    type: String,
  }
});
const Item = mongoose.model('Item', itemSchema);
module.exports = Item;
```

Security and Privacy

Using a combination of sessions, hashing and password validation, we improved security and privacy on our web application.

Session - expiry

Session

```
app.use(session({  
  ⚡ secret: 'your_secret_key',  
  resave: false,  
  saveUninitialized: true,  
}));
```

```
let timeout;  
  
function resetTimer() {  
  clearTimeout(timeout);  
  timeout = setTimeout(logout, 30000); // 30秒後にlogout関数を実行  
}  
  
function logout() {  
  //session logout if user is not active for 10 seconds  
  fetch('/logout', {  
    method: 'POST'  
  }).then(() => {  
    alert('Session timed out due to inactivity.');    window.location.href = '/login';  
  });  
}
```

```
app.post('/purchase/:itemId', ensureAuthenticated, async (req, res) => {  
  try {  
    ⚡ const user = await User.findById(req.session.user._id);  
    if (!user.purchaseHistory) {  
      user.purchaseHistory = [];  
    }  
    user.purchaseHistory.push(req.params.itemId);  
    await user.save();  
    res.json({ msg: 'Item purchased' });  
  } catch (error) {  
    console.error(error);  
    res.status(500).send('Server error');  
  }  
});
```

Ensure Authenticated

Using the below function, we ensured the user was authenticated before allowing them to use certain functions like posting or deleting.

```
function ensureAuthenticated(req, res, next) {  
  if (req.session && req.session.user) {  
    return next();  
  } else {  
    res.status(401).send('Please log in first to use these operations.');  }  
}  
  
module.exports = ensureAuthenticated;
```

Hashing

Using the bcrypt module, we hashed the given passwords

```
try {
  const existingUser = await User.findOne({ username: req.body.username });
  if (existingUser) {
    return res.render('register', { errors: ['Username already exists'] });
  }

  const hashedPassword = await bcrypt.hash(req.body.password, 10);
  const newUser = new User({
    username: req.body.username,
    email: req.body.email,
    password: hashedPassword
  });
  await newUser.save();
}
```

Dynamic Endpoints

To improve performance, scalability, and overall system reliability, we ensure that the resources are accessed or distributed in an optimal and responsive manner using dynamic endpoints.

```
app.get('/register', userController.getRegister);
app.post('/register', userController.postRegister);
app.get('/privacy-policy', userController.getPrivacyPolicy);

app.get('/', (req, res) => {
  |   res.render('home'); //to home.ejs
});
```

Viewer (Front-end)

Server-Side Rendering

Improved performance on first page load and better SEO. Also, less JavaScript is required on the client side, which reduces the load on the browser.

Example: EJS templates such as item.ejs and login.ejs retrieve data on the server side and use that data to generate HTML.

```
<body>
  <div class="navbar">
    <a href="/"><i class="fas fa-home icon"></i>Home</a>
    <a href="/items"><i class="fas fa-box-open icon"></i>View Your Items</a>
    <a href="/login"><i class="fas fa-sign-in-alt icon"></i>Login</a>
    <a href="/register"><i class="fas fa-user-plus icon"></i>Register</a>
  </div>
```

Routing to pages

```
app.get('/register', userController.getRegister);
app.post('/register', userController.postRegister);
app.get('/privacy-policy', userController.getPrivacyPolicy);

app.get('/', (req, res) => {
  res.render('home'); //to home.ejs
});

app.get('/login', userController.getLogin);
app.post('/login', userController.postLogin);
app.post('/logout', userController.postLogout);
app.get('/member', userController.getMemberPage);
//Display password reset page
app.get('/reset-password', userController.getResetPassword);
```

Improves user experience by allowing direct access to specific content and functionality via URLs.

When a user accesses a specific URL, the corresponding page or content is displayed.

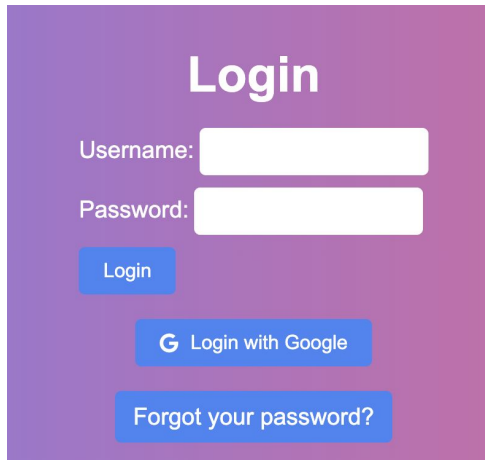
Example: Accessing /login will display the login page, /register will display the registration page, etc.

Used template engine: `express.js`

Data from databases and APIs can be dynamically incorporated into HTML, allowing for flexible page generation. It is also easy to reuse code and maintain.

Examples: `item.ejs` dynamically displays a list of products, `login.ejs` displays a login form, etc.

Buttons


A login form UI mockup on a purple background. It features the title "Login" in white. Below it are two white input fields for "Username:" and "Password:". Under the password field is a blue "Login" button. Further down is a blue button with the Google logo and the text "Login with Google". At the bottom is a blue button with the text "Forgot your password?".

Login

Username:

Password:

Login

 Login with Google

Forgot your password?

Examples: item.ejs includes a "GET" button to view product details, a "PUT" button to edit an item, a "DELETE" button to delete an item, etc. login.ejs includes a "Login" button and a "Login with Google" button

Controller

Password Validation

“GetRegister” & “postRegister”: Display the registration form and process registration, respectively.

“getLogin” & “postLogin”: Display login forms and process login.

“PostLogout”: Displays the user logout form and processes the user's logout.

GetMemberPage: Displays the member page after logged in.

Password reset function: It allows a user to request a password reset, validate the reset token, and set a new password.

Relationship between server.js and usercontroller.js

The relationship between server.js and userController is that server.js handles routing (which URL paths trigger which actions) and delegates the actual processing of those actions to methods in userController.

Microservices

Called two external APIs to display the weather and the new in japanese

[Home](#) [View Your Items](#) [Login](#) [Register](#)

Weather

Tokyo: 29.09°C

Top News

- 【随時更新】イスラエル軍 1日でガザ地区320か所以上を空爆 - nhk.or.jp
- 富山 住宅敷地内で男性がクマに襲われ搬送 現場でクマ1頭駆除 - nhk.or.jp
- 浮体式洋上風力、大量生産へ技術協力 日本・デンマーク - 日本経済新聞
- AirPods Pro（第2世代）が機能アップ。短期集中運転「iPhoneを使いこなしOS 17徹底活用術」。「通信型NCJと『会話感』（村上タクト）」 - au Webポータル
- 沢井製薬 品質確認試験を不正な方法で実施 自主回収へ - nhk.or.jp
- 43歳・熊切あさ美、太もも全開「超ミニ」美脚コーデに反響「素敵」「可愛すぎる」 - ORICON NEWS
- 宮内庁長官「陛下は気になさっておられません」... 朝野両院議長が開会式の所作ミスで陳謝 - 読売新聞オンライン
- 日本大学への国の補助金 異例の3年連続全額不交付決定 - nhk.or.jp
- 私はイスラエルとの和平を望んでいたのに...15歳の息子は射殺された：朝日新聞デジタル - 朝日新聞デジタル
- 【東京国庫映画祭】稲垣吾郎「嫌なやつに思われるかも」 新垣結衣らの「初めての表情」に期待 - ORICON NEWS
- 立憲民主党と共産党、次期衆院選の連携で合意 - 読売新聞オンライン
- 室蘭市 / 2023年（令和5年）広報むろらん11月号PDF版 - 室蘭市
- キャンパスに“大きな衝撃”国立大学教授の男（52）が腎臓病入院検査で陽性→緊急逮捕、教員を養成する「福岡教育大学」で教鞭～自宅から吸引用？のストローも押収【九州厚生局麻薬取締部】 - RKB毎日放送NEWS
- トヨタ、26日から全面的に生産再開...「パネメーカ」の爆発事故で16日から停止 - 読売新聞オンライン
- 神木隆之介&浜辺美波、東京国庫映画祭にトップバッターで登場「レッドカーペットを歩いて光栄」 - スポーツ報知
- モリカワの14本はテラー 新旧モデル混在 オリジナルアイアンも - ゴルフダイジェスト・オンライン
- 「ブレイレボ」久しぶりだな、スネーク。「メタルギア」初期作品を総産した待望の「METAL GEAR SOLID: MASTER COLLECTION Vol.1」 - 4Gamer.net

APIs for news in Japanese

```
<div class="card" id="news-section">
  <h3 class="section-title">Top News</h3>
  <ul id="news"></ul>
</div>
```

```
async function fetchNews() {
  const response = await fetch('/api/news');
  const data = await response.json();
  const headlines = data.map(article => `<li>${article.title}</li>`).join('');
  document.getElementById('news').innerHTML = headlines;
}
```

```
app.get('/api/news', async (req, res) => {
  const apiKey = 'ec7fa3239492418fb269d22f9f96ef59';
  const response = await fetch(`https://newsapi.org/v2/top-headlines?country=jp&apiKey=${apiKey}`);
  const data = await response.json();
  res.json(data.articles);
});
```

API for weather

```
<div class="card" id="weather-section">
  <h3 class="section-title">Weather</h3>
  <div id="weather"></div>
</div>
```

```
async function fetchWeather() {
  const response = await fetch('/api/weather');
  const data = await response.json();
  document.getElementById('weather').innerText = `Tokyo: ${data.temp}°C`;
}
```

```
app.get('/api/weather', async (req, res) => {
  const apiKey = '370ec73325035f836a6cdbcc22ec3181';
  const response = await fetch(`https://api.openweathermap.org/data/2.5/weather?q=Tokyo&appid=${apiKey}`);
  const data = await response.json();
  res.json({ temp: data.main.temp });
});
```

```
JS authMiddleware.te...  
! config.yml  
! itemID-config.yml  
! items-config.yml  
! login-config.yml  
! logout-config.yml  
! mainconfig.yml  
! newpassword.yml  
! purchase-config.yml  
! recommend.yml  
! register-config.yml  
! resetpassword-con..
```

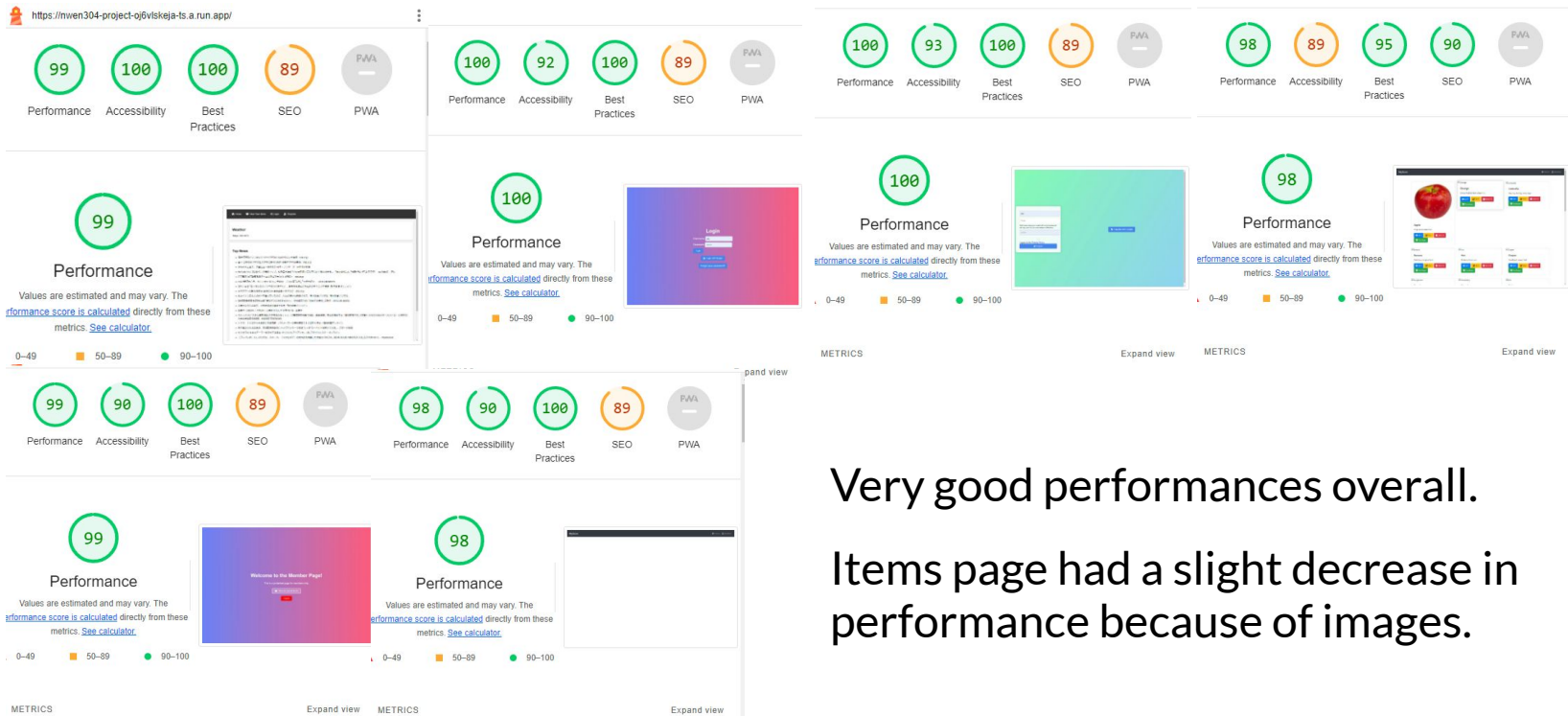
Testing

We tested the middleware using chai and performed load testing using YAML files.

Example file | Command to run `artillery run test/config.yml`

```
config:  
  target: 'https://nwen304-project-oj6vlskeja-ts.a.run.app/'  
  phases:  
    - duration: 60  
      arrivalRate: 5  
  
  scenarios:  
    - flow:  
      - get:  
        url: '/member'  
#member page
```

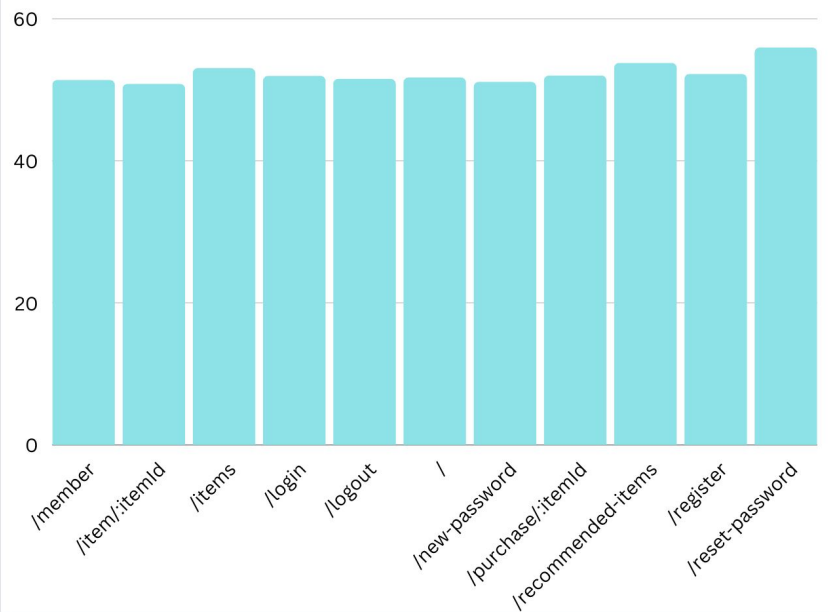
Lighthouse Performances for Rendering



Very good performances overall.

Items page had a slight decrease in performance because of images.

Bottlenecks | Request Response Times



The use of async improved our performance times and our schemas didn't contain much data.

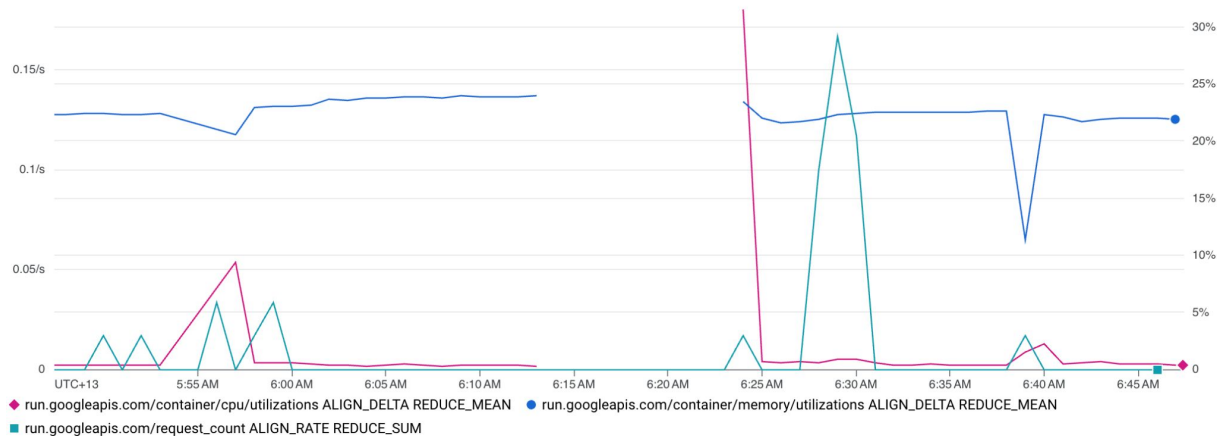
Very balanced throughout all requests.

Bottlenecks | CPU and RAM Utilization

(-) CPU Utilization

(-) RAM Utilization

(-) Mean Request
Count



No apparent spikes in CPU and RAM utilization when requests spiked

Middleware

We made “ensureAuthenticated” function that checks if the user is authenticated or not.

“ensureAuthenticated” functionality:

1. Check for req.session and req.session.user:

It checks if there's a session and a user associated with that session

2. Proceed if Authenticated:

If req.session.user exists, the middleware calls next(), which means it passes control to the next middleware function or the route handler.

3. Return Unauthorized if Not Authenticated:

If there's no req.session.user, it means the user is not authenticated. Thus, the middleware responds with a 401 status (Unauthorized) and a message prompting the user to log in.

Relationship with server.js

In server.js, the “ensureAuthenticated” middleware is imported and used as a guard for specific routes. This means that for the routes where ensureAuthenticated is placed, a user must be authenticated to access or manipulate the data.

For example:

```
app.put('/item/:itemId', ensureAuthenticated, (req, res) => {  
  
});
```

This code above from server.js, before the logic inside the PUT route for updating an item (/item/:itemId) is executed, the ensureAuthenticated middleware is run. If the user is authenticated, the route logic will be executed. If not, the user will receive a 401 Unauthorized response.

And also other routes like...

```
app.delete('/item/:itemId', ensureAuthenticated, ...);  
  
app.post('/item', ensureAuthenticated, ...);  
  
app.post('/purchase/:itemId', ensureAuthenticated, ...);
```

use this middleware. This means these operations (adding, deleting, purchasing an item) are protected, and only authenticated users can perform them.
