Team name:

Watermelons

Team members:

Jan Iverson Eligio

Joaquin Torres

Leisha Soberano-Keawemauhili (?)

ICS 491 Semester Project

Secure Development

Summer 2019

Requirements and Design

1. Program details
    - Application title: JobMelon
    - Description: A web application which connects contractors (people that want to work) with clients (people that need work done) for any type of small job or gig (e.g., housecleaning, car service, printer trouble, etc.).
    - Function requirement specification:
        - Clients will be able to create a profile from which they can list jobs with some price tag
        - Jobs will be listed (from client listings) on the homepage of the website
        - Administrators will be able to remove job listings at will
        - Contractors (i.e., anybody else) can view job listings and contact the client (instant message on the website) to confirm that they will complete the job for them
        - Once a job is complete, payment will be performed using Authorize.net to process the payment
            - The job will then be unlisted on the job listing page once the payment is accepted
2. Type of program
    i. Web Application
3. Development Tools
    - Programming language: JavaScript
    - IDE: IntelliJ

Requirements

4. Security and Privacy Requirements
    i. Public information on clients displayed/stored on the website must be kept to a minimum (i.e., only allow information such as name/company name and general job location to be displayed publicly)
        1. Clients may choose whether or not to disclose other information (i.e., job site address, phone number, email, etc.) with the contractor through instant messaging
        2. The only information stored about clients will be their name and general location (state/city)
    ii. The website will use HTTPS to facilitate all web traffic on the site to ensure encrypted communication between the user and the website
    iii. Payment information will be stored and encrypted in the database
    iv. User passwords will be hashed/salted in the database such that a database breach will not breach passwords
        1. Passwords must meet secure requirements (at least 1 uppercase letter, at least 1 lowercase letter, at least 1 special character, at least 1 number, and at least 15 characters total)
    v. Security flaws/issues will be kept track of using the Github Issues feature (this allows us as the developers to track issues, assign them to somebody, and fix them then tie them to specific commits to our code)
5. Quality Gates (or Bug Bars)
    i. Privacy

    <u>End-User Scenarios</u>

End users refer to the clients and contractors (i.e., users of the application).

Critical

a. Lack of user controls
    i. Collection and storage of user data (e.g., location) without the ability to stop the collection and storage of this information
b. Lack of data protection
    i. Collection and storage of personal information without the ability for users to see/correct information within the database which stores this information
c. Improper use of cookies
    i. Sensitive information stored in a cookie is not encrypted/secured
d. Insufficient legal controls
    i. Transmission of data to a third party which has not signed a contract and is not disclosed within the terms of use

Important

e. Lack of user controls
    i. Collection and storage of non-essential user data (i.e., data that is not necessary to perform the purpose of the application)
f. Improper use of data
    i. Sensitive user information transmitted/stored on the site or sent to a third party is not

necessary to perform the purpose of the application

Moderate

g. Lack of data protection
    i. Data stored temporarily has no method to prevent unauthorized access to the data during transmission or storage
h. Lack of data management
    i. Data stored on site does not have a retention policy
i. Lack of user controls
    i. Collection and storage of non-essential user data (i.e., data that is not necessary to perform the purpose of the application)
j. Improper use of data
    i. Sensitive user information transmitted/stored on the site or sent to a third party is not necessary to perform the purpose of the application

Low

k. Lack of notice/consent
    i. User metadata is stored but hidden without notice to the user

Administration Scenarios

Administrators refer to administrators acting to ensure a quality user experience (e.g., administrators delete job listings and user accounts which violate the terms of use for the application)

Critical

l. Lack of administrative controls
    i. Collection and storage of sensitive user data as an administrator without explicit user consent as described in the terms of use

Important

m. Lack of administrative controls
    i. Collection and storage of non-sensitive user data as an administrator without explicit user consent as described in the terms of use

Moderate

n. Lack of administrative controls
    i. Lack of method for accidental collection and storage of any user data as an administrator

ii. Security

Critical

a. Privilege escalation
    i. User obtains more privilege than authorized
b. Database (SQL) injection
    i. A user (or attacker) deletes or modifies data stored in the databases on the server which hosts the website

Important

c. Privilege escalation
    i. User obtains more privilege than intended (authorized to, but shouldn’t be)

d. Denial of Service
    i. The functionality of the website is affected through means of a denial of service attack
e. Spoofing
    i. A user (or attacker) “spoofs” or masquerades as a user other than their own (unintentionally or intentionally)

6. Risk Assessment Plan for Security and Privacy

i. Roles and Responsibilities
    1. Roles will be designated for chief responsibility of security and privacy, though it will be the job of everyone to make sure all risks are assessed

ii. Privacy
    1. Determine Privacy Impact Rating
        a. The rating will represent the amount of risk the application will possess in terms of privacy and will be assessed on a P1, P2, and P3 scale. P1 being the highest.
        b. The software will be assessed by the following behaviors:
            i. P1 - Stores personally identifiable information
            ii. P1 - Provides experience that targets children or is attractive to children
            iii. P1 - Monitors user continuously
            iv. P1 - Installs other new software, changes file type associations, home or search page
            v. P2 - Transfers anonymous data
            vi. P3 - None of the above

2. Make any adjustments to software design or requirements that could lower the impact rating
3. Agree on design that agrees with privacy guidelines
4. Analyze design to identify personally identifiable information stored or transferred in P1 scenarios. Elaborate on privacy aspects of the software in an analysis. Scenarios include:
    a. Describe the personally identifiable information stored or data transferred
    b. Describe reasons the personally identifiable information is required for software
    c. Describe all other software to be installed or changes to file types, home page or search page
    d. Describe notice and consent experiences
    e. Describe how users can gain access to public disclosure
    f. Describe how you will prevent unauthorized access to personally identifiable information
5. Create a draft of privacy disclosure
6. Determine who to contact in case of privacy incident

iii. Security

1. Identification
    a. All critical assets of the software will be determined. Then, a list of all sensitive data that is created, stored, or transmitted by the assets will be compiled and a risk profile will be created for each one.
2. Assessment
    a. Create an approach to assess the security risks for the critical assets.
3. Mitigation

a. Create a mitigation approach and enforce security controls for each of the risks identified.

4. Prevention
    a. Use tools to minimize threats and vulnerabilities.

Design

7. Design Requirements

i. Information stored in the database will be kept to a minimum such that user privacy is not breached in the event of a data breach

ii. Data will be separated into separate tables (“collections” in MongoDB) in a database

1. A collection for end users (endusers)
    a. username (string), the user’s username
    b. password (string), the user’s hashed password

    Note: User passwords will be hashed/salted in the database such that a database breach will not breach passwords

    c. ccNumber (int), the user’s credit card number
    d. ccSecurityCode (int), the user’s credit card security code
    e. ccExpiryDate (string), the user’s credit card expiration date (MM/YY)

    Note: Payment information will be encrypted and stored for 30 days or until the payment is confirmed and received by the contractor, at which point it will be automatically erased according to the data retention policy

i. PCI compliance must be upheld

2. A collection for administrator users (adminusers)
    a. username (string), the user's username
    b. password (string), the user's hashed password

    Note: User passwords will be hashed/salted in the database such that a database breach will not breach passwords

3. A collection for client job postings (listings)
    a. jobName (string), a short description of the job
    b. price (Decimal128), how much the user lists the job for
    c. cityName (string), what city the job is located in
    d. stateCode (string), what state the job is located in (e.g., HI, CA, AZ, etc.)
    e. description (string), a longer description of the job with more details
    f. datePosted (date), when the job was posted
    g. owner (string), the user who posted the job
    h. complete (boolean), whether or not the job is complete
4. A collection for messages between users (messages)
    a. to (string), the user which the message was sent to
    b. from (string), the user which the message was from
    c. message (string), the message that was sent

    Note: Instant messaging data between clients and contractors will be kept for 60 days according to the data retention policy

    d. dateSent (date), when the message was sent

iii. User information can only be accessed by the owner and administrators with consent to access it

iv. Security flaws/issues will be kept track of using the Github Issues feature (this allows us as the developers to track issues, assign them to somebody, and fix them then tie them to specific commits to our code)

v. The user interface will go through extensive testing before and after each version of the software is released to the public

8. Attack Surface Analysis and Reduction

i. Unregistered visitors will only be able to view job listings on the web app as is. Clients will have the ability create jobs for a certain hourly rate and to confirm it with the contractors. Contractors have the ability to accept certain jobs. Administrators will have elevated privilege and be able to destroy jobs made my clients at will.

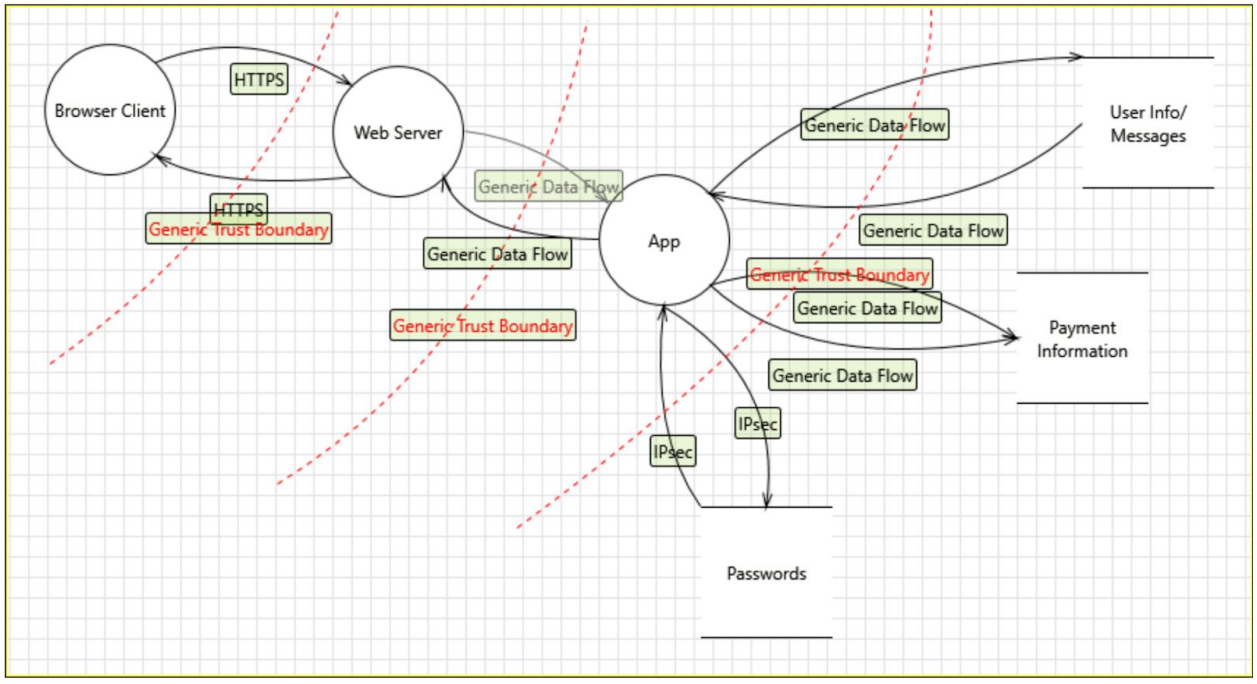
ii. Paypal: Known Security Vulnerabilities

1. CVE-2017-6215
    a. paypal/permissions-sdk-php is vulnerable to reflected XSS in the samples/GetAccessToken.php verification_code parameter, resulting in code execution.
2. CVE-2017-6213
    a. paypal/invoice-sdk-php is vulnerable to reflected XSS in samples/permissions.php via the permToken parameter, resulting in code execution.
3. CVE-2017-6099
    a. Cross-site scripting (XSS) vulnerability in GetAuthDetails.html.php in PayPal PHP Merchant SDK (aka merchant-sdk-php) 3.9.1 allows remote

attackers to inject arbitrary web script or HTML via the token parameter.

4. CVE-2013-7202
    a. The WebHybridClient class in PayPal 5.3 and earlier for Android allows remote attackers to execute arbitrary JavaScript on the system.
5. CVE-2013-7201
    a. WebHybridClient.java in PayPal 5.3 and earlier for Android ignores SSL errors, which allows man-in-the-middle attackers to spoof servers and obtain sensitive information.
6. The rest of the known security vulnerabilities here

9. Threat Modeling



Implementation

GitHub Online Repository: https://github.com/uh-watermelons/jobmelon

README File: https://github.com/uh-watermelons/jobmelon/blob/master/README.md

1. Approved tools

Tool	**Minimum Required Version and Switches/Options**	**Optimal/ Recommended Version and Switches/Options**	**Comments**
Source Code Management System	Git/GitHub	Most recent version	JobMelon Repository
JavaScript IDE	IntelliJ IDEA 2019 Ultimate		IntelliJ IDEA Ultimate is offered for free to student accounts registered on JetBrains
idealU-2019.1.3.exe	Version 2019.1.3	Most recent version	
Static Analysis	SonarLint Version 4.0.2.3009	Most recent version	Offered by IntelliJ IDEA as a plugin
JavaScript Programming Language	ECMAScript 6	Most recent version	ECMAScript is a subset of JavaScript and is setup through the settings in the IntelliJ IDEA IDE
Database	MongoDB Atlas	Most recent version	Cloud database

	Version 4.0+	(4.0.10)	
Node.js	Version 10.16.0 LTS	Most recent version	JavaScript run-time environment for server side Javascript code execution
npm	Version 6.9.0	Most recent version	Package manager for Node.js
React		React 16.8	User interface

2. Deprecated/Unsafe Functions

- IntelliJ IDEA
  i. JavaScript expression closures
     - Alternative: function expression or arrow functions
  ii. JavaScript for-each-in loop
     - Alternative (ECMAScript 6): for-of statement
- Node.js
  i. node --debug
     - Alternative: node --inspect
  ii. Using non-string values for process.env
     - Alternative: use strings for process.env
  iii. All util.is*() functions
     - Alternative: validate variables/objects differently
- React
  i. HTML Tag to center text:
     - Alternative: text-align
  ii. HTML Attribute to specify background image: background
     - Alternative: background-image

iii. React - componentWillMount function
- Alternative: componentDidMount

iv. React - componentWillReceiveProps function
- Alternative: getDerivedStateFromProps

v. React - componentWillUpdate function
- Alternative: componentDidUpdate
- Alternative: getSnapshotBeforeUpdate

- MongoDB
  i. Using the $where operator for filtering (e.g., through collection find method), NoSQL injection
     - Alternative: validate/verify user input (disallow certain characters, limit length)

3. Static Analysis
   - SonarLint is a static code analyzer for as an integrated development environment extension that helps developers detect and fix quality issues while writing code. SonarLint is offered by IntelliJ IDEA as a plugin. SonarLint supports ECMAScript 6 and React JSX. SonarLint in IntelliJ IDEA detects issues in Java, JavaScript, Python, Kotlin, Ruby and PHP.
     i. Additional features of SonarLint includes:
        - Analyze a set of files.
        - Exclude specific files and issues.
        - Enable more rules, or mute some rules.
        - Find logs.
        - Sync rules, issues and exclusions to SonarCloud or SonarQube.
   - SonarLint is very similar to ESLint, which was used as a JavaScript linter for ICS 314. However, setup was really easy and loading time was quicker compared to ESLint. Code that does not follow the basic coding standards set with SonarLint is easily detectable by the colored squiggly

lines/highlighting and a red exclamation symbol in the corner. When there is an error, there is a light bulb that you can select with possible fixes that can be automatically applied. This tool is just an easy, all-around tool with no problems so far.

Verification

1. **Dynamic Analysis**
    - We have chosen iroh.js as our dynamic analysis tool for our application. The tool is used to view runtime call tree graphs, type checking, code quality, test cases, and for code visualizations. As of today (6/16/19), we have not used the tool for testing yet as our progress has been limited to our front-end.
2. **Attack Surface Review**
    - As of today (6/16/19), there have not been any changes, updates, patches, or new vulnerabilities reported in any of the approved tools that we have listed.

Verification II

1. **Fuzz Testing**
    - Input validation
        i. Our application has four forms. Two forms were for registering a user and logging in. In our registration form, we adhered to the password requirements of at least 1 uppercase letter, at least 1 lowercase letter, at least 1 special character, at least 1 number, and at least 15 characters total. We put a cap of 30 characters for the password. Emails must also be valid. To test this route, we used Postman to send HTTP requests to our server. We tested our API routes with emails that were invalid and passwords that did not

meet the requirements. This confirmed that our back-end form validation for registering users worked correctly. The other two forms were for public viewing. There was potential for a script to be injected into the form fields. Before sending the data to the server, we used validate.js to first sanitize the input. We made sure to make the validator send an error back to the client if these characters were present in the data: “&”, “<”, “>”. We deemed these characters to be in unnecessary to use in any part of our application. When testing this API route, the validator correctly sent an error back to the client.

- Testing application robustness with “Monkey Testing”
    i. We used the node module, gremlins.js, to check the robustness of our application. The module simulates random user actions. A horde of “gremlins” can be dispatched to click anywhere in the window, enter random data in forms, or move the mouse over elements that do not expect it. The goal was to trigger JavaScript errors and make the application fail. In our first attempt, there were not enough errors generated to crash the application. We believe the reason is because the styling we used was minimal and we only fetched data from our database when necessary. Also, profile images were kept at a maximum of 50x50 pixels. To make this more robust, we will implement pagination in our home page when displaying job listings to limit the amount of listings in a page.
- Security misconfiguration
    i. This was less of a “hack” and more of a security concern. Our ESLint module (which does static analysis) was printing out misconfigurations in our front-end configuration to the web console (in the openly available “Inspect Element” tool that most modern browsers have). Although there was nothing sensitive at the time of

testing, there exists the potential for sensitive information about how the site works or if there are any exploitable misconfigurations to be displayed through this console.

To mitigate this, we could disable the errors from printing to the console using the ESLint rules to disable it (no-console: “off”).

2. **Static Analysis Review**
    - After reviewing our code with our static analysis tool (ESLint/SonarLint), we found a lot of unused variables and imports, unterminated statements (just needed semicolons), and errors in our HTML formatting. Some of the warnings were also because of formatting with our function calls (SonarLint throws warnings when they are separated onto different lines).
    - We also found some errors in the code, like where it found that we didn’t have text in a certain area on one of our forms, and another where it found that the images we tried to import were missing.
    - Some parts highlighted through these tools, however, turned out to be false positives. Some would say “cannot resolve <variable>”, even though they existed and the code would compile without error. Therefore, these types of errors were ruled out.
    - In summary, static analysis helped us clean up our code substantially to lessen the confusion in the components of our application.
3. **Dynamic Review**
    - After reviewing our code with our dynamic analysis tool, the JavaScript analysis tool “Iroh”, we didn’t find any vulnerabilities/errors or code corrections during runtime.
    - We used the staging tool in Iroh to wrap all of our applicable scripts (mainly, our server side JS) so that we could analyze each part of the code in a stepwise/textual fashion (individual function calls and object properties broken down). This allowed us to see the in-depth details for

some of the framework components that we used that we previously did not understand.

- Although not very critical, we also used Iroh to evaluate the performance of our web app (for loading pages/functions) of which loading times were negligible as expected (mainly due to the lack of production data being loaded on the client side).

Release

Incident Response Plan

Privacy Escalation Team

Note: Due to a lack of human resources, we have overlap between roles.

Escalation Manager (Jack Torres) - responsible for delegating responsibility and tasks across the privacy escalation team to ensure the completion of the incident response process.
Legal Representative (Jack Torres) - responsible for handling legal concerns throughout the incident response process by working with a legal team to determine the best courses of action.
Public Relations Representative (Jan Eligio) - responsible for handling public relation concerns throughout the incident response process through consistent and timely response on public social media and press events.
Security Engineer (Jan Eligio) - responsible for “bridging the gap” (i.e., filling in the technical details) between the technical and legal aspects of incident response. Also responsible for documenting processes and procedures throughout the incident response process to ensure similar incidents do not recur.

Emergency Contact

In the case of an emergency (your data has been breached, your account has been compromised, etc.), you may contact us via email at info@jobmelon.com. A representative will respond to you within 1-2 business days. If the emergency is urgent, please contact the local police.

Incident Response Procedures

It is of paramount importance that incident response procedures be in place in the event of a security incident such that it may be dealt with quickly, effectively, and with the least amount of damage to all parties affected. The procedures outlined below will help to determine the root cause of an incident to prevent recurrence of such incidents in the future.

1. Establish team responsibilities
    a. This step is preferably performed before an incident occurs, but is necessary to establish a base to start the process from.
    b. The purpose of this step is to define who does what during this procedure (i.e., who will be the security engineer, who will be the public relations representative, etc.).
    c. At JobMelon, we have established these roles (as detailed above in the **Privacy Escalation Team** section).
2. Identify timeline expectations and goals
    a. The purpose of this step is to define when certain goals are expected to be met. These goals include:
        i. Determining who is affected
        ii. Determining who to notify
            1. Preparing PR statements
        iii. Determining the source of incident
            1. Refactoring code as necessary
            2. Stopping affected services immediately
            3. Releasing patches
        iv. Determining legal repercussions if applicable
3. Document process
    a. Documentation of this process will aid in training staff (current and potential) of the implications and responses for such incidents as well as provide an audit trail for who was responsible for what during this process.
4. Evaluation
    a. After the process is complete (i.e., software has been patched, all affected parties have been notified), the incident response team should evaluate the course of action in the future when moving past the incident and for repeat incidents.

Final Security Review

After careful review of the software, we at JobMelon can confidently say that our software should receive the **Passed FSR with exceptions** grade according to the Final Security Review (FSR) process detailed by Microsoft here.

The reason for this judgement is due to the following:

- Threat Model
    - Our application meets all requirements as defined in the Threat Model, with an exception:
        - In the current version of the software, the method of connection to the MongoDB database that we use to store our user information, including sensitive information like credit card numbers and user account passwords, is currently not as secure as it would be in a “real-world” product.
        - If this product were to be released in the “real-world”, we would take the connection string and encryption keys out of the source code and store them into environment variables on the client side such that the client cannot infiltrate the database.
- Static Analysis
    - Our application meets all requirements as defined in the Static Analysis section, meaning that there are no potentially insecure settings/configurations present in our code that could be detected with our static analysis tools.
- Dynamic Analysis
    - Our application meets all requirements as defined in the Dynamic Analysis section, meaning that there are no insecure runtime events that could be detected with our dynamic analysis tool.
- Quality Gates/Bug Bars
    - Our application meets all requirements as defined in the Quality Gates/Bug Bars section (some parts were even omitted in the final release of the project, see the Future Features below).

Certified Release & Archive Report

Final release of the JobMelon application

https://github.com/uh-watermelons/jobmelon/releases/tag/v1.0

Summary

Version v1.0

Features

- User accounts
  - Users have their own persisting profiles from which they can register and login with. User profiles store information such as name (first name required only, last name optional), email, and payment information (optional). Users have different permissions, with admin users having the ability to moderate the application by being able to remove job postings if necessary (as detailed in our Terms of Use).
- Session management
  - Users are automatically logged out after exiting the website or after 1 hour of inactivity, which means that sessions are secured for users of the website.
- Secure storage
  - We are storing our sensitive information (credit card numbers and passwords) in encrypted strings within our database so they are secured and not susceptible to hacking.
- Job listing management

    - Job listing management refers to the ability to create, delete, and modify job listings. Users are able to manage their own job listings, but not other users job listings.
- Privacy Policy & Terms of Use
    - We have included privacy policies and terms of use on the site so that users know the terms of usage of the site while they use it.

Future development plans

- Payment processing
    - We currently are storing payment information but have no means of processing payment due to our time constraints and us not wanting to actually process payment (because of money constraints). Our plan was to have multiple methods of processing payment, either through a third party like PayPal or through our own implementation.
- In-app messaging
    - We wanted to implement in-app messaging such that clients and contractors could communicate quickly and easily without having to take the conversation to email or over the phone (unless they decided to), but we weren't able to implement it due to time and resource constraints.
- In-app notifications
    - We wanted to implement in-app notifications for any security bulletins, notifications of updates to the software or policies, and for notifications like when somebody is interested in your job (as a client) but

weren't able to implement it due to time and resource constraints.

- HTTPS implementation
    - In a production application, we would've wanted to run the app over HTTPS such that the traffic to and from the server would be encrypted and secured.
- Search functionality
    - We wanted to implement the ability to search for jobs (and therefore tag jobs with certain keywords) but weren't able due to time and resource constraints.

Usage of the program

- Technical Notes
    - How to install and use the application

      1. Clone the repository from GitHub
      To do so, you can use GitHub Desktop or by using `git` via the command line with the following command:
      - `git clone https://github.com/uh-watermelons/jobmelon.git`

      2. Install NodeJS
      The latest version of NodeJS needs to be installed in order for this application to run
      - You can download it here: https://nodejs.org/en/

      3. Install NodeJS server-side packages
      - In the command line, change directory to `../jobmelon/`
      - Install all the NodeJS packages needed for this application by typing `npm install`

      4. Install NodeJS client-side packages
      - In the command line, change directory to `../jobmelon/client/`
      - Install all the NodeJS packages needed for this application by typing `npm install`

5. Start the application
- In the command line, change directory to `../jobmelon/`
- Start the application by typing `npm run prod`

6. Open the application
- Browse to the application at http://localhost:3000/
- Note: After starting the client, your browser should automatically load the site