Smile Seeker

"Discover Your Perfect Smile, One Click Away."

Description:

“Smile Seeker is your ultimate destination for finding top-quality dental care with ease and convenience. Our platform connects clients with trusted dentists in their area, helping them discover personalized dental solutions tailored to their needs. Whether you're seeking routine check-ups, cosmetic enhancements, or specialized treatments, Smile Seeker simplifies the process by providing comprehensive profiles of dentists, including services offered, pricing, locations, and patient reviews. With user-friendly search features and seamless appointment scheduling, Smile Seeker empowers individuals to take control of their dental health and find the perfect smile they've always wanted. Join us on Smile Seeker and embark on your journey towards a brighter, healthier smile today.”

Technologies:

Front-end:

- **HTML/CSS:** These are essential for structuring and styling my website's user interface.
- **Bootstrap:** Bootstrap can be a great addition to streamline the development process and ensure the website is responsive and visually appealing across different devices.
- **React.js:** React.js is a powerful JavaScript library for building dynamic user interfaces. It can enhance the interactivity and performance of the website.

Back-end:

- **Python with Flask:** Python is a versatile and beginner-friendly language, and Flask is a lightweight and flexible web framework. Flask provides the necessary tools for handling routing, HTTP requests, and database interactions.

Database:

- **MariaDB:** MariaDB is a popular open-source relational database management system, which is compatible with MySQL. It offers robust features for storing and managing structured data. Additionally, since I am using Python for the back-end, Flask integrates well with SQLAlchemy, a Python SQL toolkit and Object-Relational Mapping (ORM) library, which can simplify database operations and enhance security.

Challenges:

Describe the problem the Portfolio Project is intended to solve:

The "Smile Seeker" project aims to address several challenges in the dental care industry:

- **Lack of Information:** Many people struggle to find comprehensive information about dentists, including their services, prices, and locations, making it difficult for them to make informed decisions about their dental care.
- **Limited Accessibility:** Traditional methods of finding dentists, such as word-of-mouth referrals or online directories, may not always provide convenient access to a wide range of dental services or dentists in the area.
- **Time and Effort:** Searching for a dentist that meets one's specific needs can be time-consuming and labor-intensive, involving multiple phone calls, appointments, and visits.
- **Geographic Constraints:** People living in remote or underserved areas may face challenges in accessing quality dental care due to limited availability of dentists in their locale.

By creating a centralized platform like "Smile Seeker," individuals can easily find and compare dentists based on their services, prices, and locations, saving time and effort while ensuring they receive the dental care they need.

Explain what the Portfolio Project will not solve:

While "Smile Seeker" aims to streamline the process of finding and selecting a dentist, it does not address every aspect of dental care or solve all the challenges in the industry. Some limitations include:

- **Quality of Care:** "Smile Seeker" provides information about dentists and their services but does not guarantee the quality of care provided by each dentist. Users may still need to rely on other sources, such as reviews or personal recommendations, to assess the quality of a dentist's work.

Explain who the Portfolio Project will help and/or who the users will be:

The primary beneficiaries of the "Smile Seeker" project include:

- Individuals seeking dental care: Users can search for dentists based on their specific needs, preferences, and location, allowing them to find a dentist that suits them best.
- Dentists and dental practices: Dentists can create profiles on "Smile Seeker" to showcase their services, prices, and locations, helping them attract new patients and expand their client base.

Is this project relevant or dependent on a specific locale?

While "Smile Seeker" is designed to be accessible to users worldwide, its relevance may vary depending on the availability of dentists and dental services in a specific locale. In areas with a high concentration of dentists or urban centers, users may have more options to choose from and may find "Smile Seeker" helpful for comparing dentists and services. However, in rural or underserved areas with limited access to dental care, "Smile Seeker" may have a more significant impact by connecting users with available dentists and promoting awareness of dental services in the area.

Technical Risks:

- **Scalability:** As the user base grows, there may be challenges in scaling the application to handle increased traffic and data volume. The potential impact could include slow performance, downtime, or system crashes.

  Safeguard/Alternative: Implement scalable architecture and cloud infrastructure (such as AWS or Google Cloud Platform) to accommodate growing demand. Using load balancing, caching mechanisms, and database optimization techniques to improve performance.

- **Security Vulnerabilities:** The website may be susceptible to security threats such as data breaches, SQL injection, or cross-site scripting (XSS) attacks, compromising user data and damaging the platform's reputation.

  Safeguard/Alternative: Follow security best practices, such as input validation, encryption, and parameterized queries, to mitigate common vulnerabilities. Regularly update software libraries and frameworks to patch security flaws. Implement user authentication, authorization, and session management mechanisms to protect sensitive data.

- **Third-party Dependencies:** Dependency on external libraries, APIs, or services may introduce risks related to compatibility issues, service disruptions, or changes in third-party terms and conditions.

  Safeguard/Alternative: Evaluate third-party dependencies carefully before integration. Monitor dependencies for updates and changes. Have backup plans or alternative solutions in place in case of service disruptions.

Non-Technical Risks:

- **Reputation Damage:** Negative user experiences, poor reviews, or publicized incidents (such as data breaches) could harm the platform's reputation and credibility, leading to decreased user trust and adoption.

  Strategy: Prioritize user satisfaction and trustworthiness by providing transparent and reliable services. Actively solicit and address user feedback, resolve issues promptly, and maintain open communication channels with users. Implement reputation management strategies to monitor and respond to online feedback and reviews.

- **Market Competition:** The dental services market may be competitive, with established players or alternative solutions offering similar services. Difficulty in differentiating the platform or attracting users could impact the project's success.

Strategy: Conduct market research to understand competitors and identify unique value propositions or niche markets. Focus on providing superior user experience, innovative features, or specialized services to differentiate the platform. Develop effective marketing and branding strategies to attract and retain users.

Infrastructure:

Process for Branching and Merging in the Project Repository:

- Feature Branches: Developers create feature branches from the main branch (e.g., master or main) to work on specific features or fixes. Branch names should be descriptive and prefixed with the issue or feature they address (e.g., feature/search-functionality).
- Commits: Developers make incremental changes and commit them to their feature branches with clear and concise commit messages.
- Pull Requests (PRs): Once a feature is complete, developers create a pull request to merge their changes back into the main branch. PRs should include a description of the changes, relevant screenshots or documentation, and any necessary tests.
- Merge: After approval, the feature branch is merged into the main branch using a merge commit or a squash merge, depending on the project's version control workflow.

Strategy for Deployment:

- **Continuous Integration/Continuous Deployment (CI/CD):** Set up a CI/CD pipeline to automate the build, test, and deployment process. CI tools like Jenkins, Travis CI, or GitHub Actions can automatically build and test your application whenever changes are pushed to the repository. Deployment can be triggered automatically after successful tests, ensuring a streamlined and efficient release process.
- **Deployment Environment:** Maintain separate environments for development, staging, and production. Develop and test new features in the development environment, conduct user acceptance testing in the staging environment, and deploy stable releases to the production environment for public access.
- **Rollback Plan:** Have a rollback plan in place in case of deployment failures or unforeseen issues. Automated or manual rollback mechanisms can revert to the previous stable version to minimize downtime and mitigate risks.

Populating the App with Data:

- Manual Entry: Initially, I can manually input sample data or create dummy accounts to showcase the application's features and functionality during development and testing.
- Seed Data: Use database seeding techniques to populate the database with test data. Define seed files containing predefined data records (e.g., dentists, services, prices) and run seed commands to insert them into the database.

Testing Tools, Automation, and Processes:

- Unit Testing: Unit tests to validate the behavior of individual components, functions, or modules. Use testing frameworks like Jest (for React.js) or pytest (for Python) to automate unit tests and ensure code correctness.
- Integration Testing: Conduct integration tests to verify the interaction and compatibility between different modules or components of your application. Tools like Selenium or Cypress can automate

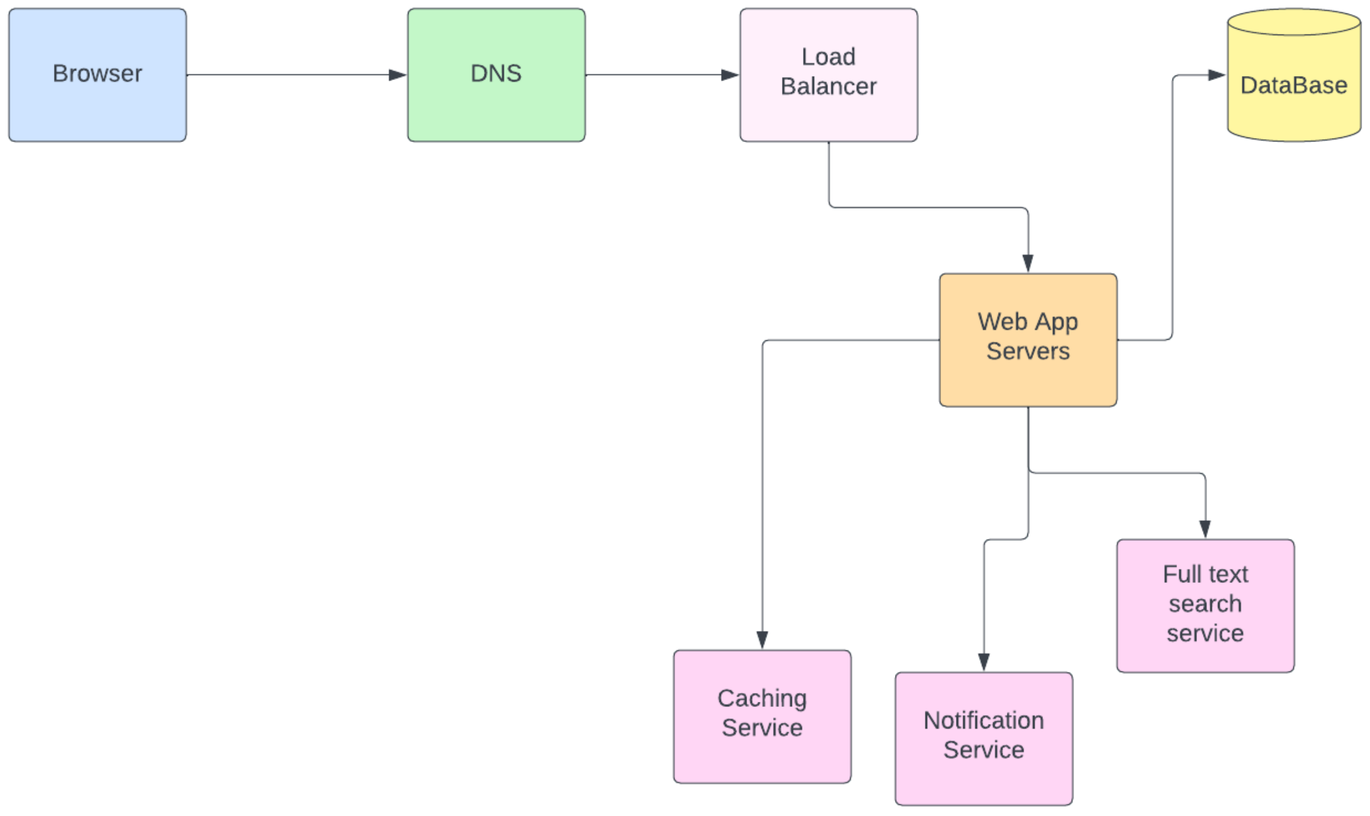
browser-based testing for frontend interactions.

- End-to-End Testing: Perform end-to-end tests to simulate real user scenarios and validate the entire application workflow. Tools like Selenium, Puppeteer, or TestCafe can automate end-to-end testing across multiple layers of your application.
- Manual Testing: Supplement automated testing with manual testing to explore edge cases, user experience, and usability issues that may not be covered by automated tests. Develop test plans, test cases, and test scripts to guide manual testing efforts and ensure comprehensive test coverage.

Existing Solutions:

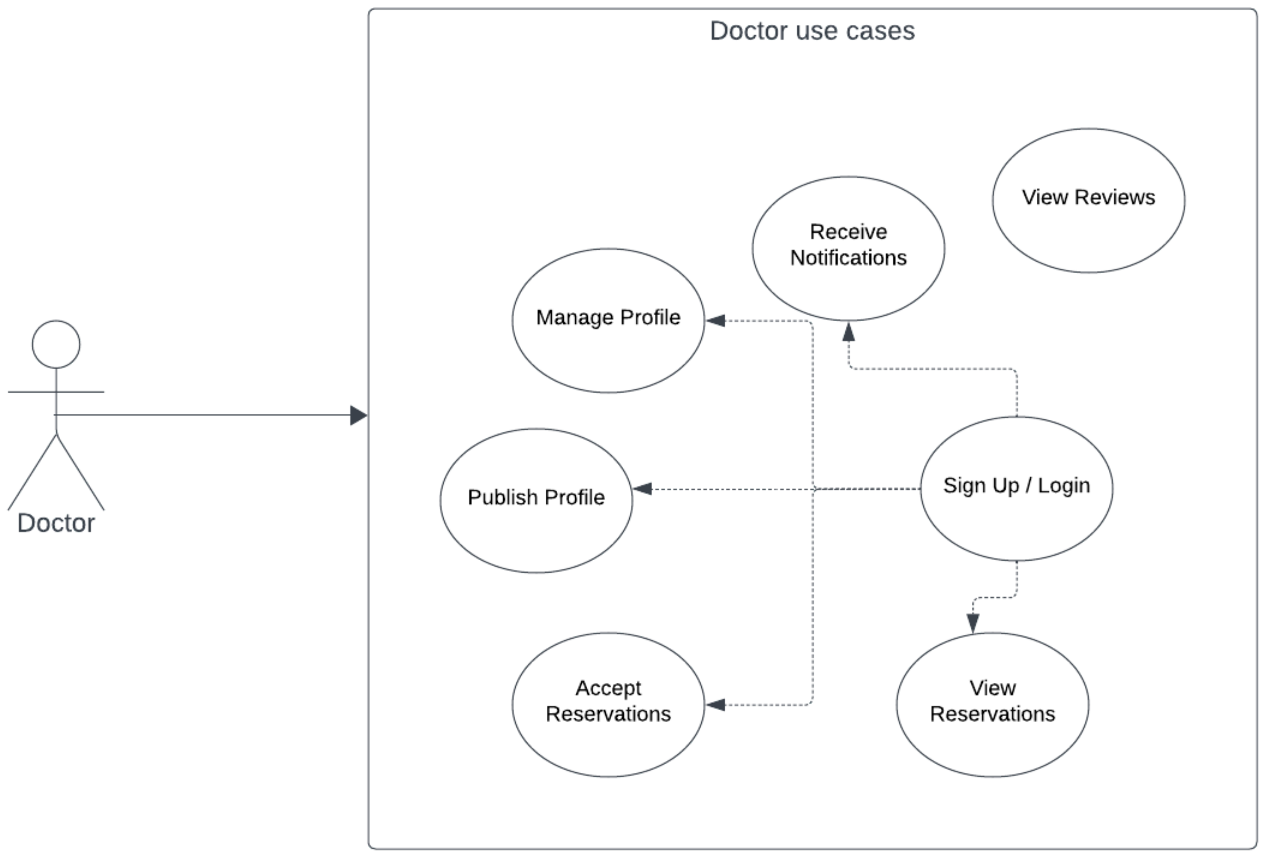
- Google Maps: Google Maps is a widely-used mapping service that allows users to search for businesses, including dentists, in their area. Users can view business listings, read reviews, and get directions to the dentist's office. However, Google Maps primarily focuses on mapping and navigation and may lack specialized features for dentists, such as detailed service listings, appointment scheduling, or price comparisons.

Architecture:

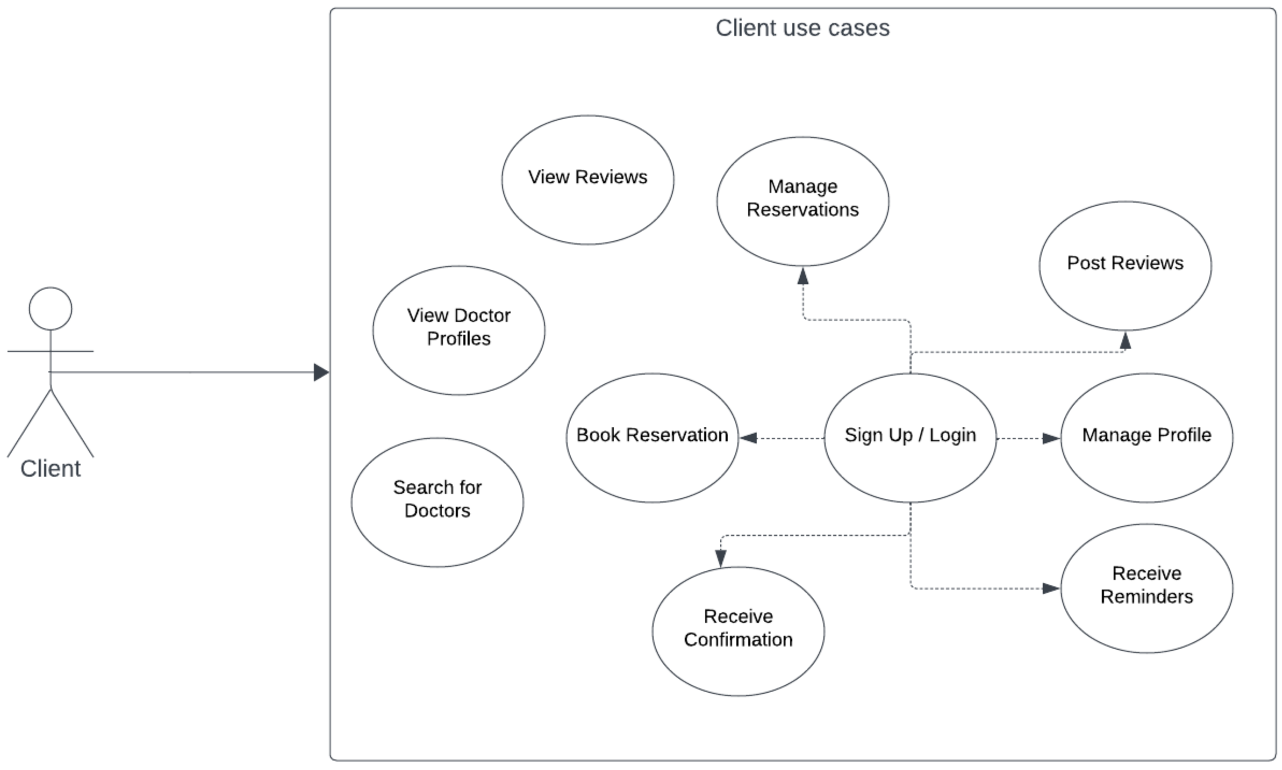


User cases:

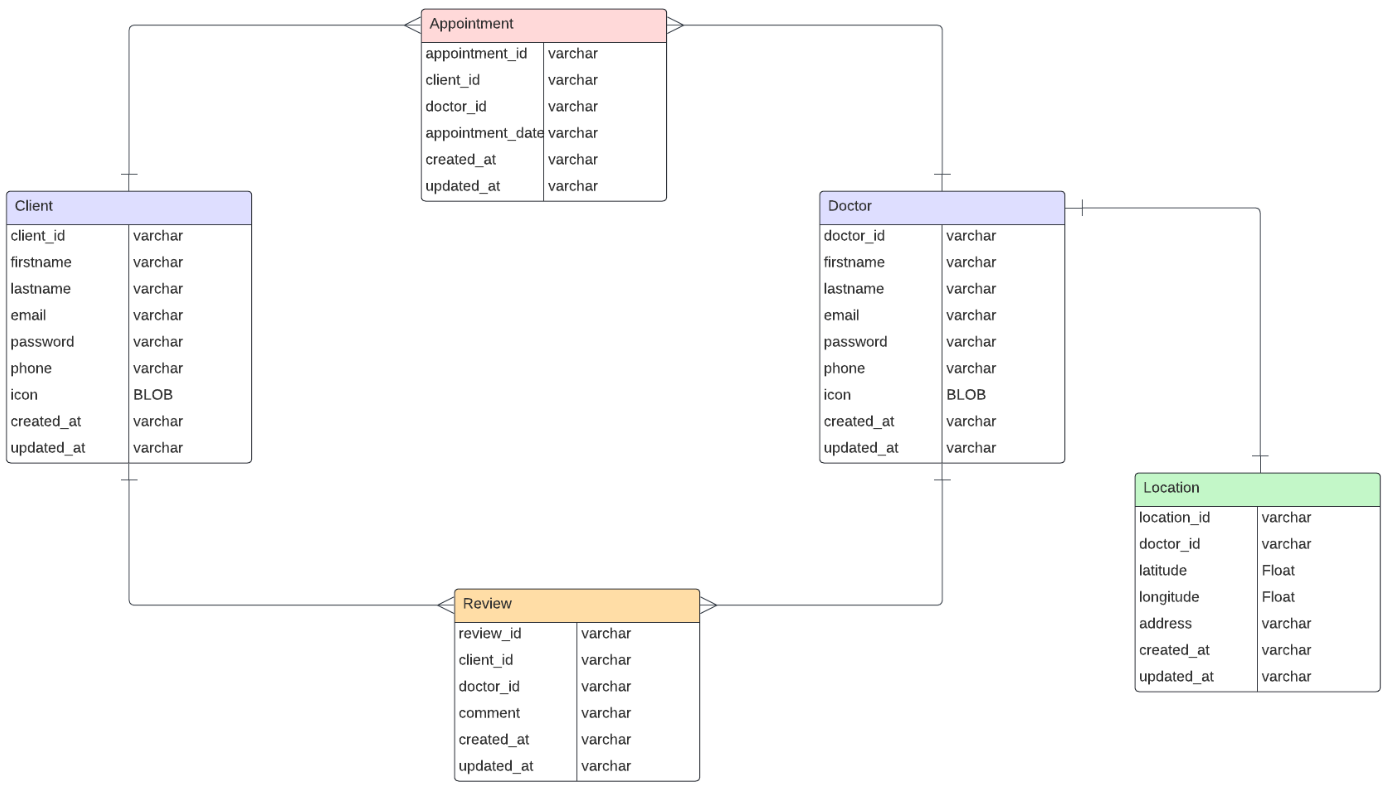
- **Doctor**


Doctor use cases
Doctor
View Reviews
Receive Notifications
Manage Profile
Publish Profile
Sign Up / Login
Accept Reservations
View Reservations

- **User:**


Client use cases
Client
View Reviews
Manage Reservations
Post Reviews
View Doctor Profiles
Book Reservation
Sign Up / Login
Manage Profile
Search for Doctors
Receive Confirmation
Receive Reminders

Entity Relationship Diagram (ERD):



APIs:

/api/doctors	/api/doctors/<doctor_id>
• **Get**: the list of all the doctors. • **Post**: create a doctor object	• **Get**: get the doctor object using ID • **Put**: update the doctor object • **Delete**: delete the doctor object

/api/users	/api/user/<user_id>
• **Post**: create a user object	• **Get**: get the user object • **Put**: update the user object • **Delete**: delete the user object

/api/doctors/<doctor_id>/reviews	/api/reviews/<review_id>
• **Get**: get the reviews about a doctor • **Post**: create a review about a doctor	• **Get** a review by id. • **Put** update a review. • **Delete**: delete a review

/api/appointments	/api/doctors/<doctor_id>/appointments
• **Get** all the appointments.	• **Get** all the appointments to a doctor.

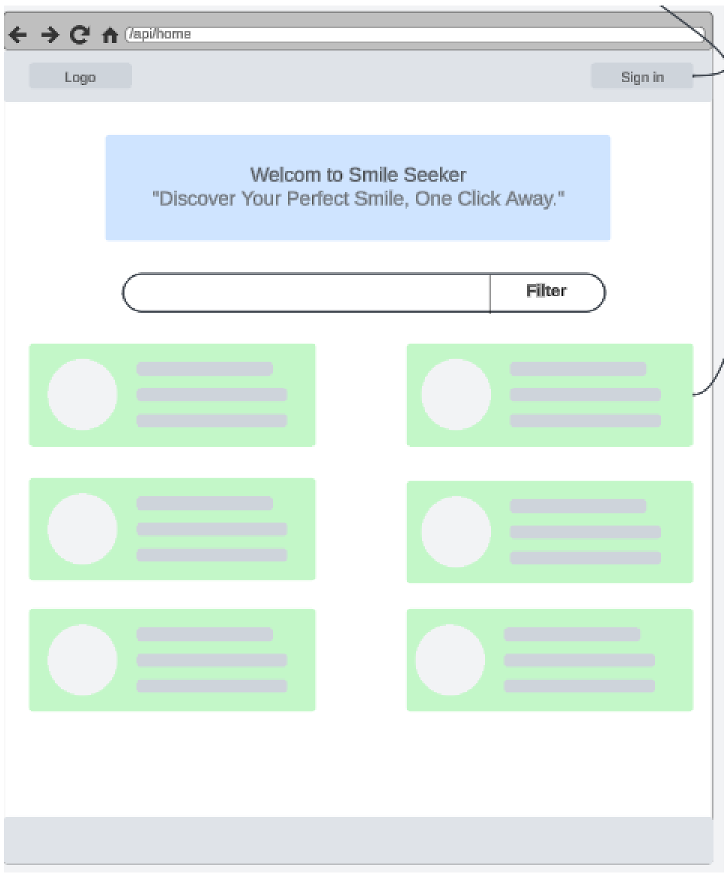
/api/users/<user_id>/appointments	/api/users/<user_id>/appointments/<appoint_id>
• **get** all the appointments of a user. • **Post** makes an appointment.	• **Delete**: delete an appointment object

3rd party APIs:

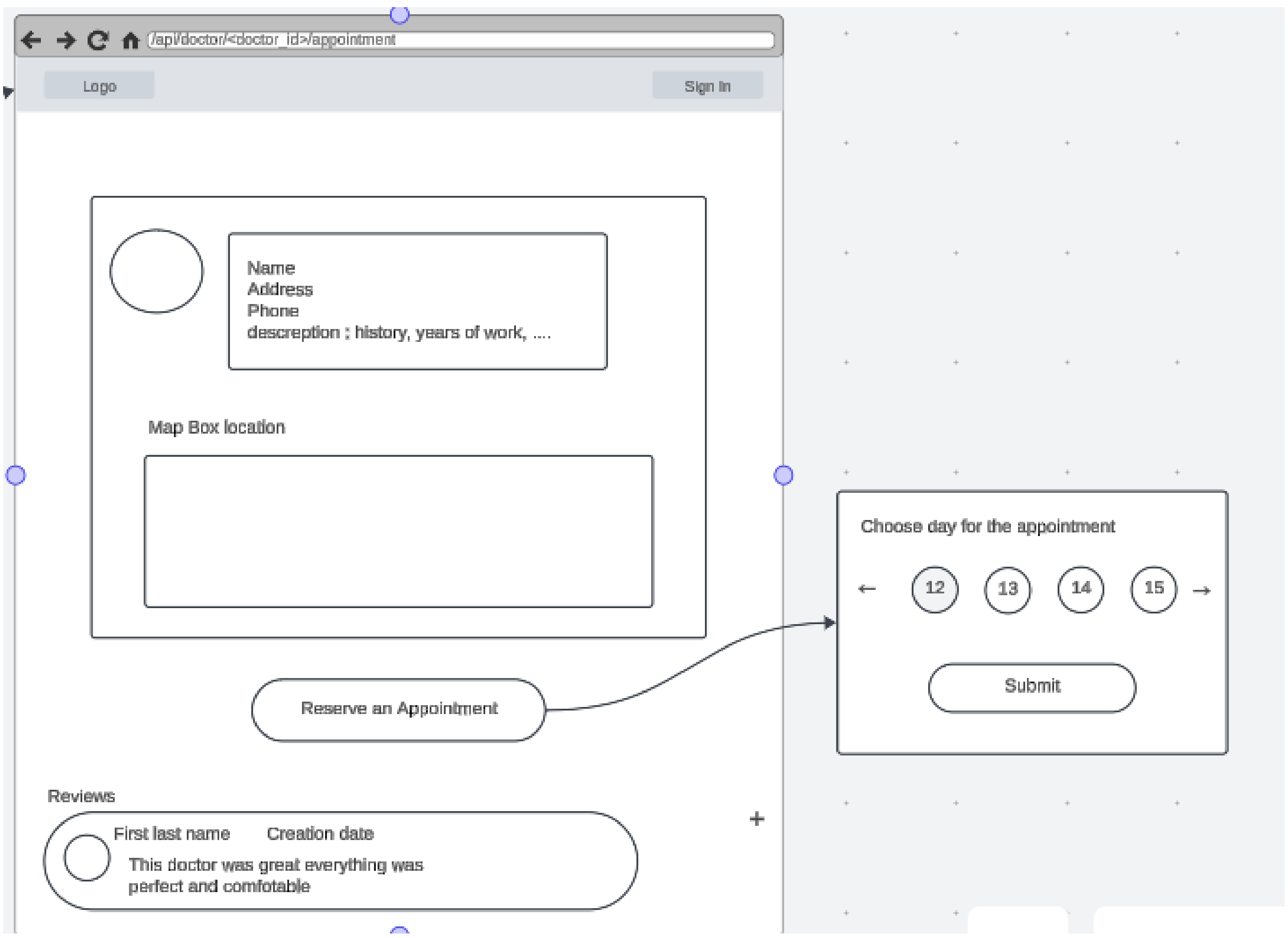
Map Box Maps: https://api.mapbox.com/directions/v5/{profile}/{coordinates}
GET: get the directions to the doctor address

Mockups:

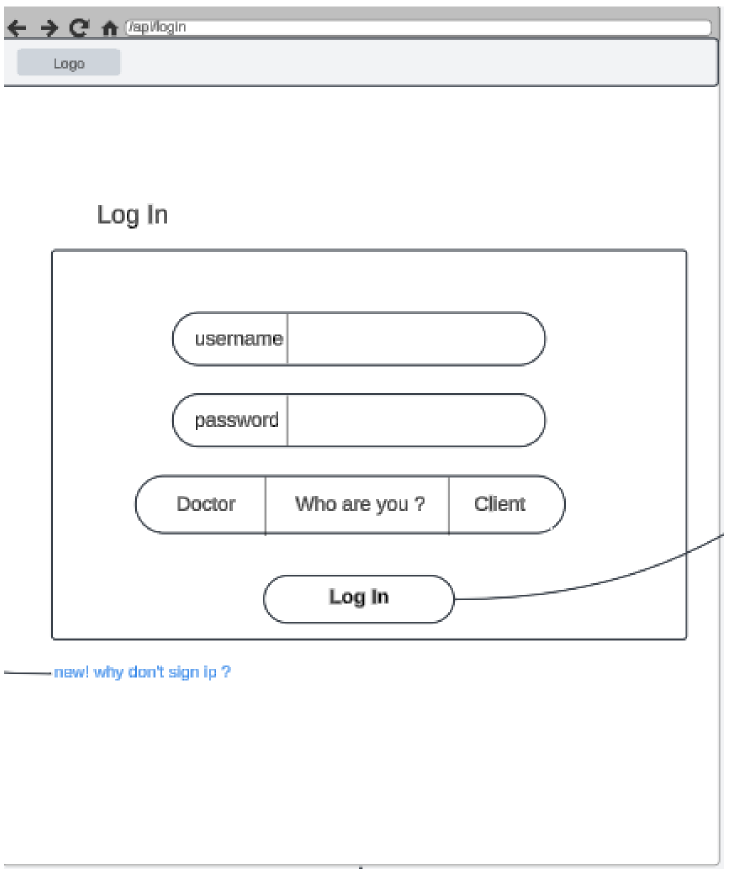
- **Home page:**



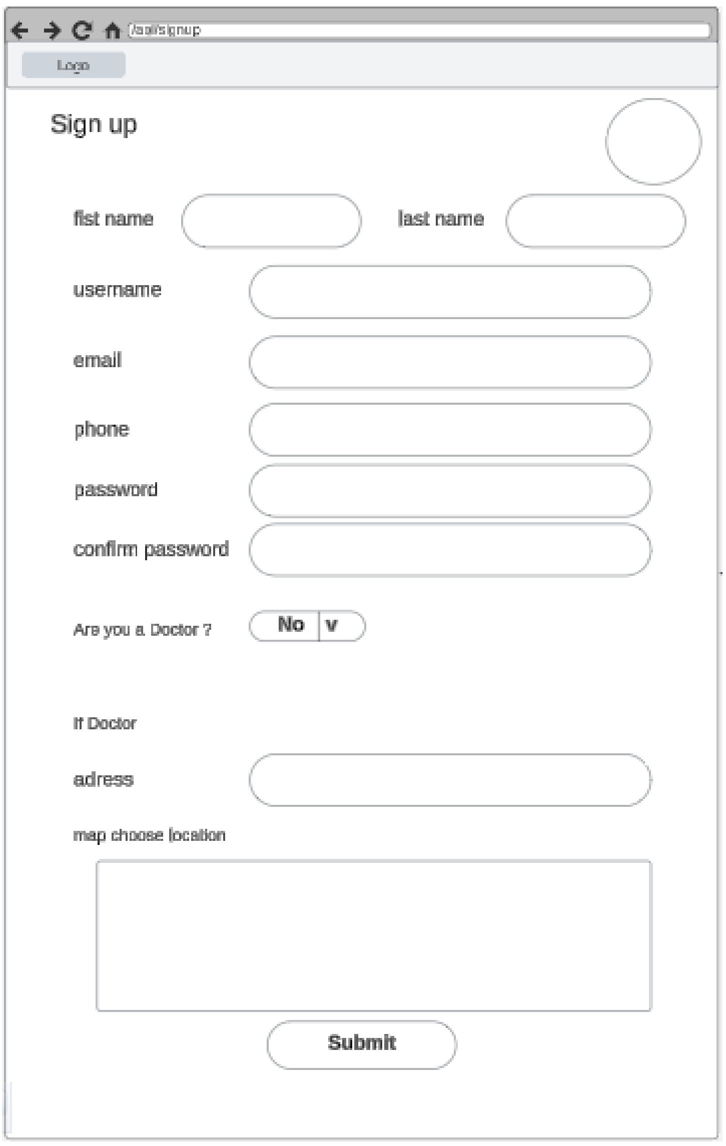
- **Select a doctor and reserve an appointment:**


/api/doctor/<doctor_id>/appointment
Logo
Sign In
Name
Address
Phone
descreption ; history, years of work, ...
Map Box location
Reserve an Appointment
Choose day for the appointment
12
13
14
15
Submit
Reviews
First last name
Creation date
This doctor was great everything was perfect and comfotable

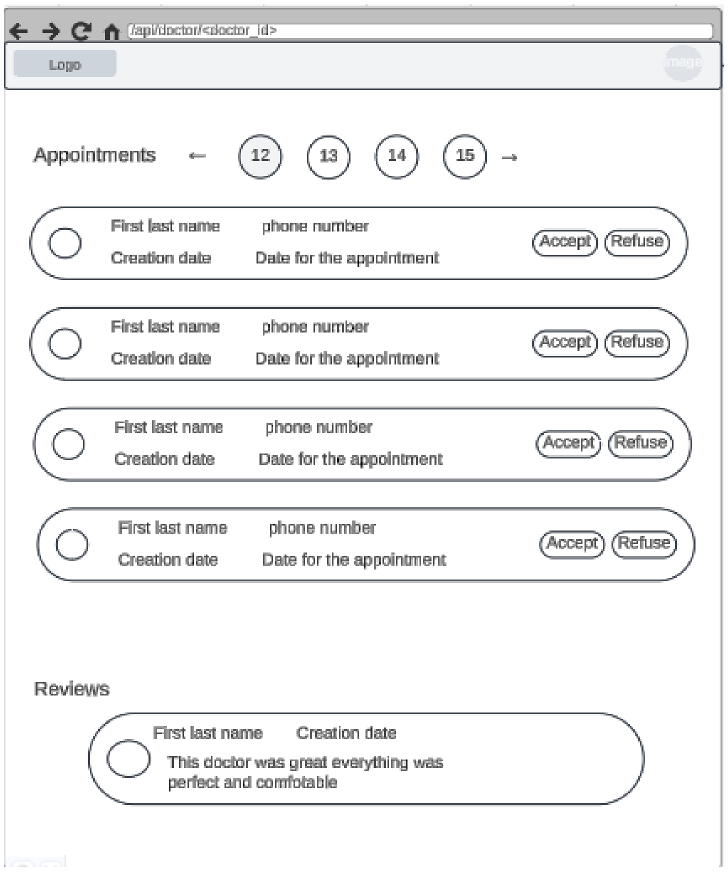
- **login**


/api/login
Logo
Log In
username
password
Doctor
Who are you ?
Client
Log In
new! why don't sign ip ?

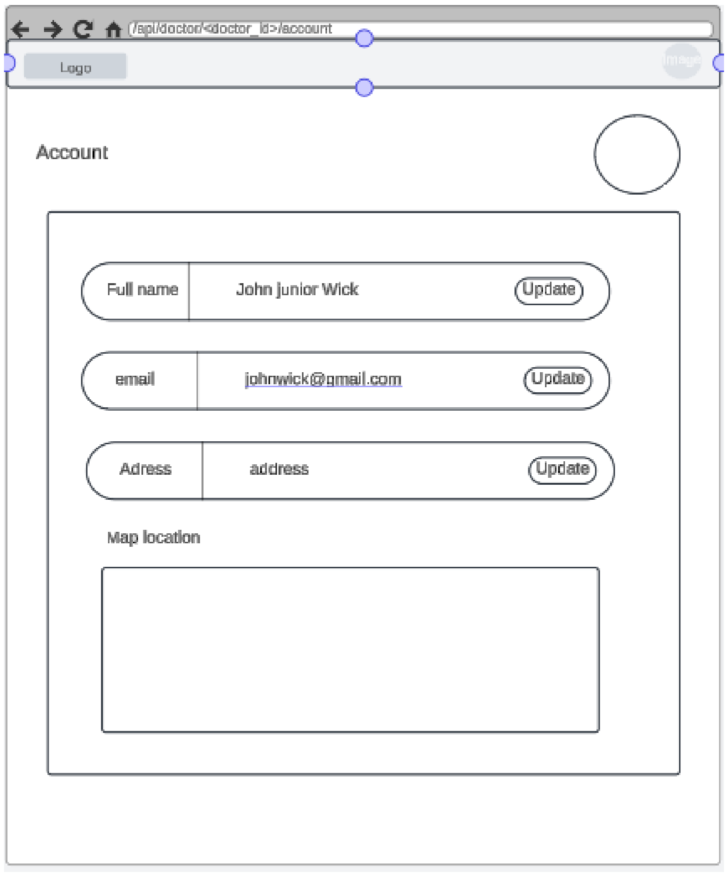
- sign up page:


/api/signup
Logo
Sign up
fist name
last name
username
email
phone
password
confirm password
Are you a Doctor ?
No
v
If Doctor
adress
map choose location
Submit

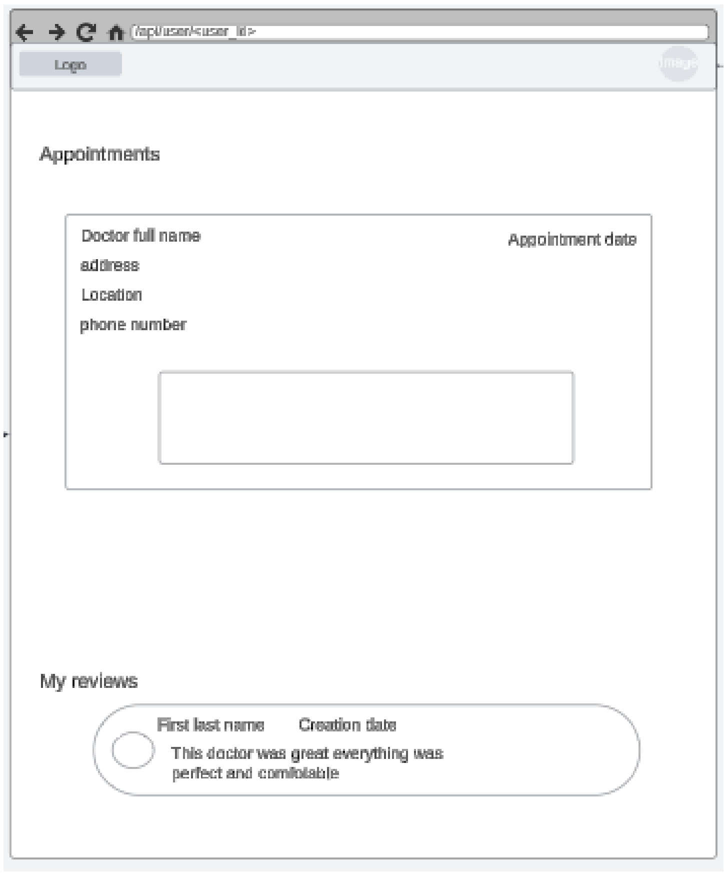
- doctor page:



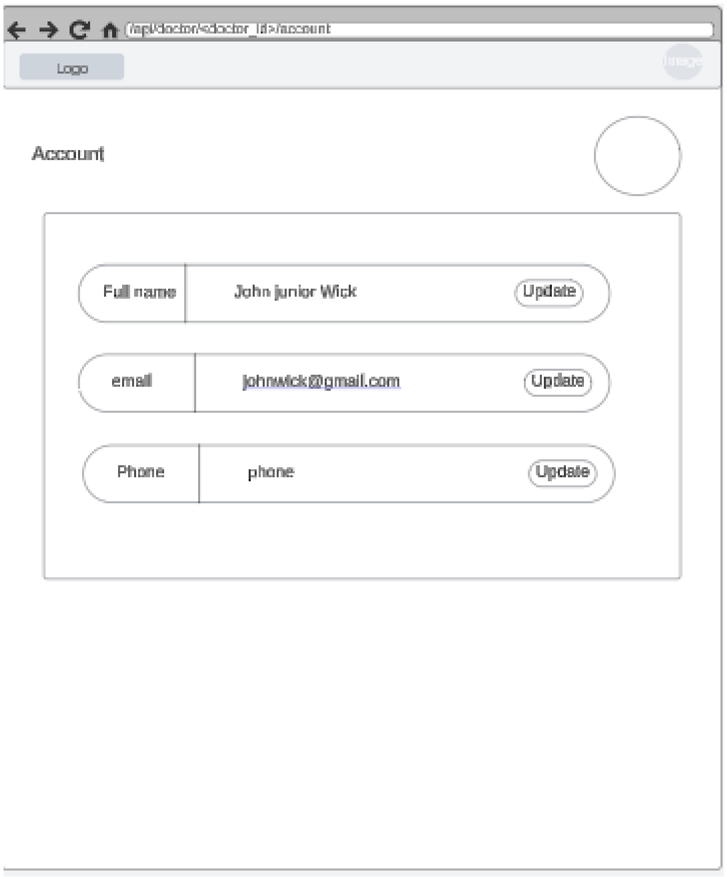
- doctor profile:



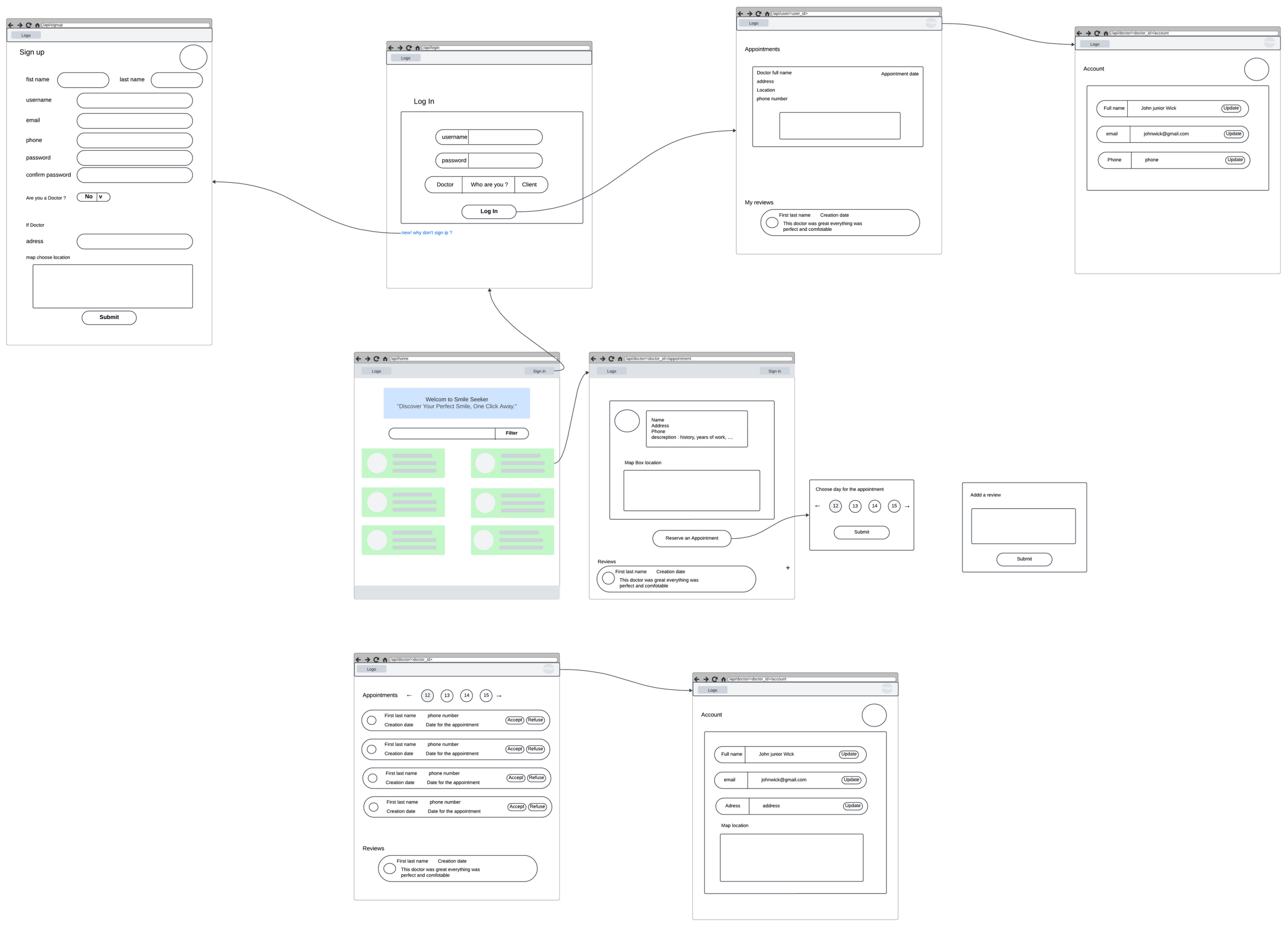
- user page:



- user profile



- Overview:



- link for the website wireframe:

https://lucid.app/lucidspark/2b759848-5ec6-43bd-a756-bc4da1d27bca/edit?invitationId=inv_4d2553c2-de16-4905-aeaf-7e48755e2c7d