Caso #1, 30%

Groups of 3.

Description

20minCoach is a real-time coaching platform that connects people with experts across multiple fields—such as health, psychology, law, mechanics, programming, cloud services, arts, agriculture, and more—through on-demand 20-minute video sessions. Users can describe their need by text or voice, review available coaches' profiles, ratings, and specialties, and instantly connect once the coach accepts the request. Coaches manage their availability directly in the app, while ratings influence both their reputation and earnings. The business model is based on flexible packages that allow users to purchase a set number of coaching sessions, making expert guidance affordable, accessible, and time-efficient.

20minCoach: A Day in the Life of Users and Coaches

Imagine waking up on a Monday morning and facing a challenge: your car makes a strange sound, and you're not sure what it means. Instead of spending hours searching online, you open 20minCoach, an app designed to instantly connect people with professional coaches across dozens of areas—mechanics, psychology, art, law, cloud services, agriculture, and more.

As a user, you simply type or record a quick voice note explaining what you need: *"I think my car is making a noise from the engine, can someone help me understand what it could be?"* In seconds, the system searches for available coaches, prioritizing those closest to you. You see a mechanic's profile: photo, professional background, customer reviews, and areas of expertise. Everything looks good, so you hit *Connect.* The mechanic receives a notification, confirms availability, and a 20-minute video call starts right away. During this call, you show the noise on camera, describe what you hear, and the mechanic guides you on what to check and whether a repair shop visit is urgent.

The session ends after exactly 20 minutes, giving you focused, personalized advice without overcommitting your time or money. You rate the coach, leaving a short review. That rating directly contributes to the coach's earnings and reputation within the platform.

From the coach's perspective, the day looks different. Take Maria, a professional painter who uses her mornings to update her profile on 20minCoach, uploading new images of her work and writing about her teaching style. She sets herself as *Available* before starting her day. Later, while having coffee, she receives a

notification: someone nearby is looking for help with watercolor techniques. She reviews the request, accepts, and within moments is in a video call. For 20 minutes, she provides guidance, tips, and even small live demonstrations. When the session ends, Maria checks her earnings, notes her updated rating, and toggles her availability off before moving on with her day.

The Business Model 20minCoach works with prepaid packages, making access to expert help affordable and flexible. For example:

- *Starter Package*: $19.99 per month for 2 coaching sessions.
- *Pro Package*: $59.99 per month for 8 coaching sessions.

This model allows users to plan ahead while giving coaches a consistent flow of opportunities to earn, all within a simple and structured 20-minute framework.

Case resolution overview

Your task is to design the frontend architecture for the 20minCoach platform. This is a comprehensive software design project that will require research, decision-making, and detailed documentation. You will work on creating a scalable, maintainable, and efficient frontend system.

You must generate documentation for the proposed system design. The document should cover the key aspects that need to be considered in the design, but the structure and format are left to the discretion of the working group. The focus should be on clarity and completeness of content rather than presentation style, ensuring that the information is easy to follow and logically organized.

The documentation should be written intelligently and concisely, avoiding unnecessary filler descriptions or generic “AI-generated” text that does not add value. The objective is to provide clear and actionable details that directly support the development effort. Each section must address essential design elements such as scope, modules, data, processes, and technical requirements without unnecessary complexity or vague explanations.

A practical way to validate the quality of the design specification is to assess whether a programmer could take the document and begin implementing the project without needing to ask any questions to the team. The documentation must be precise enough to remove ambiguities and complete enough to serve as the only reference needed for development.

Core Tasks to Perform

1. Proof of concepts to develop, required to refine the design

Proofs of concept are exercises that involve programming or the use of specific tools and must be carried out to clarify design decisions.The purpose of a proof of concept is not to deliver a final product but to reduce uncertainty in the design process. It provides concrete evidence that guides decision-making, ensuring the chosen approach is reliable and can be successfully integrated into the overall system.

Frontend Source code

- include in the repository the src folder with the complete project structure
- such structure must match the designed frontend architecture
- all PoCs source code and requested classed in the design must be store in this folder in the properly
- all files, templates, source as guidance must be save in the correct layer folder where its going to be use in the final implementation

Testing

- Design testing strategy and technology, focus on unit testing
- Implement three unit tests for two different classes
- Add the testing running scripts and proof that test can pass and/or fail
- Add instructions to the dev team of how to add new unit tests and how to run them

UX & security

1. Generate a prototype screen using AI for the search and result of coaches. This must match all the technologies selected.
    - Use an AI tool to generate an initial layout. Before choosing the tool, confirm that it supports your desired language and that the prototype can be tested with your chosen UX testing tool.
2. Select a UX testing tool to evaluate your screen designed.
    - Examples: Maze, Useberry, Lookback, Optimal Workshop, make your the tool provides heat maps.
    - Upload your prototype to the tool.
    - Define at least 2 tasks for users to complete (e.g., "search for a coach specialized in fitness", "accept the suggested coach").
    - Recruit 3–5 participants not enrolled in the course to run the UX test

- Share the test URL with the participants to perform the UX test, record the evidence
- Gather metrics like task completion rate, time on task, error rate depending and heat map
- Document all the results and the UX test evidence

3. Add authentication and authorization
    - Create a simple login screen with email, password, and a "Sign in" button. (this screen is automatically created by some security providers)
    - Select two actions in the screens of your prototype to be assigned to specific role permissions.
    - In the management console of the selected auth provider implement the follow configuration
    - Create two roles: (role names are just placeholders)
        - BasicUser: can perform only Action A.
        - PremiumUser: can perform both Action A and Action B.
        - Add the application and APIs permissions
        - Assign the permission to the roles
        - Create test users and assign them to roles.
        - Enabled Two Factor Authentication
        - Integrate the login screen into the functional prototype
    - Runing your app test with both roles using the specific users and their MFA (BasicUser and PremiumUser).
    - Verify that the UI renders only the actions the logged user is authorized to perform, is highly suggest to use server side rendering approach

--

Design document

1. Technology Research and Selection

- Research modern frontend frameworks and libraries
- Compare technologies such as React, TailWind, Vue, Angular and similar for this specific use case
- Evaluate state management solutions such as Redux or Mutex
- Research real-time communication technologies (WebRockets, WebRTC, Notification Services)
- Select testing frameworks and tools
- Choose styling methodologies and tools
- Choose a linter and unit test technology
- Document your technology choices with justification

2. N-Layer Architecture Design

- Design a layered architecture for the frontend application, layers are specify below
- Define clear responsibilities for each layer
- Establish communication patterns between layers
- Ensure separation of concerns and maintainability
- Document this before the architecture diagram

3. Visual Components Strategy

- Develop a component organization strategy, this might be lead by the technology choose
- Design how to achive a reusable component library structure, those are steps for the developers
- Create a component development workflow based on the technology selected, those are steps for the developers
- Establish component testing methodology, this is not theory, are steps for the developers

Detailed Layer Design Requirements

The following items are layers that you should consider to include in your application frontend architecture, all those items must match in the project folder structure.

Visual components

- Design a component hierarchy based on the selected technology
- Specify how reusable UI components will work
- Decide accessibility standards
- Design the responsive guidelines within code examples of the practices that the dev team must to follow
- Object design patterns might be required

Controllers

- Design controllers for business logic mediation
- Do not forget clarify the hook-based connectors in the controllers
- Handle user input validation and processing
- Consider a strong usage of dependency injection

Model

- Design the most important model classes, specially for those required for the key design patterns in your solution
- Implement model validation documenting with an example what validator are you going to use and how to use it, provide developers with instructions of how to create more validators

Middleware

- Implement request/response interceptors
- Create middleware for permissions validation
- Design and implement an error handling middleware
- Design and implement an log events middleware
- All of them are required to be code and provide implementation templates or examples in the source code to guide software engineers
- Object design patterns might be required

Business

- Study the theory of domain driven design and what technology is available in the choosen language to achive such paradigm
- Design the classes holding the business logic
- Create reusable business logic services
- Implement domain-specific rules and validation
- Design business logic testing strategy
- Provide implementation templates or examples in the source code to guide software engineers
- Object design patterns might be required

Proxy/Client/Services

- Design API client abstraction layer, providing templates of how APIs are going to be integrated into the future
- Create the client for the security layer, this is going to be functional code

Background/Jobs/Listeners

- Design listeners for real-time updates
- Design periodic data refresh mechanisms

- Provide implementation templates or examples in the source code to guide software engineers
- Object design patterns might be required, pub/sub

Validators

- Correlate this section with the model design
- Provide at least one example of the validator and proper guidelines as explained in model

DTOs

- Design Data Transfer Object interfaces if need it
- Explain how and when DTOs are going to be required
- Create a transformation template and example to be use between API calls and frontend models, this can be a middleware as well
- Provide proper instructions

State management

- Select and design the state management solution
- Include this on either the architecture diagram or class diagram

Styles

- Choose and design how CSS or styles are going to be manage
- Design the responsive rules of the design and how the responsiveness is going to be test
- Design an strategy for dark/light mode support and how to test it
- Provide clear instructions to developers

Utilities

- Desing the utilities layers modeling with one example is enough
- Singleton pattern might require

Exception Handling

- Design and implement in code an standard way to handle exceptions
- Make sure the exception handling use the logging layer

- Make sure pure errors are not raise to upper levels of the code and produce a friendly message
- Object design patterns might be required
- Make sure all code templates and examples use it

Logging

- Design structured logging system using strategy pattern to allow multiple logging providers
- Implement a general Logger class
- Make sure all code templates and examples use it

Security

- Design authentication and authorization layers
- This is going to be result of the authorization PoC and the Client layer

Linter Configuration

- Select a linting tool
- Define code style rules and conventions
- Include the linter in the project and document guidelines
- Include the linter rules file adding a custom rule of your desire

Build and Deployment Pipeline

- Design build process for different environments
- Create development, staging, and production builds in the configuration files
- Create deployment documentation guidelines in the readme.md
- Add environment variable files
- Add pipeline for runing unit test
- Document instructions for developers on how to run the app, run the test and the deployment

Deliverables

1. Repository Structure

- Create a GitHub/GitLab repository
- Initialize with proper readme.md for the whole design documentation

- Include all the design components, terms, aspects in the readme.md
- Add diagrams folder for architecture visuals, embedding such images into the readme.md
- Include code examples where appropriate or links to code files or configuration files
- Add the complete project structure into the src folder and including the proof of concepts made

2. Documentation

- Create comprehensive readme.md document
- Include all sections outlined in this project brief in the desire format, work smart
- Provide clear explanations and justifications
- Use diagrams and visual aids where helpful
- Include code snippets and examples

3. Architecture Diagrams

- Create N-Layer architecture diagram
- Include object design with design patterns applied and labeled
- Clearly label all components and their relationships
- Document design patterns usage in the diagram
- Specify class responsibilities and interactions mostly when interacting with specific patterns
- The full architecture diagram must be a pdf perfect readable
- You can create one or two diagramas (architecture and classes)
- Last date for questions to the professor: Monday 22nd, september
- Last date to commit: Saturday 27th, September
- Commits are going to review to validate every group member participation along the 3 work weeks, penalties might apply

---

Class Activities:

1. Research which AI tool would be most convenient for generating the prototype and that provides access to the source code for further development.
2. Create a paper prototype of the screen for searching a coach and the screen for reviewing the selected coach.
3. Determine the following: a) AI provider to generate the prototype b) Technology stack to be used for developing the frontend c) Security provider for the integration

Present the three points above along with the prototype screens at the end of the class.

In case #1, in the section "Detailed Layer Design Requirements", detail how your group is going to document this, emphasis in what is going to be the output and the document structure.