Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: Coupl

High-level Design Report

Cankat Anday Kadim, Rüzgar Ayan, Ege Türker, Emre Derman, Kamil Kaan Erkan

Supervisor: Cevdet Aykanat
Jury Members: Shervin Arashloo and Hamdi Dibeklioglu

High-level Design Report

December 24, 2021

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

1. Introduction

Coupl is a mobile application that will create a safe environment for users to meet with new people in-person only in a matter of minutes. Event organizers will make their events available to the application and the users will join the events by scanning the QR codes provided in these event locations. The possible events include musical concerts, parties, art exhibitions and many more. Coupl will then match the users in the same event, based on their preference from the current participant pool and their history of event and matches in the application.

1.1. Purpose of the system

Coupl as a system/service aims to be an innovative alternative to the currently used dating and meeting services. By matching the users that are already participating in the same event and also with similar histories/profiles, Coupl aims to match the users of very similar interests/tastes. Compared to alternatives such as online swipe-based dating apps which only provide chatting functionality, this application will provide a much more dynamic experience with an in-person meeting.

1.2. Design goals

1.2.1. Usability / User Friendliness

- Event recommendations and suggested matches need to give reliable results to users.
- The application will be easy to use in order to attract the attention of the wide age range of the users.
- The application should be able to show upcoming events.
- Interfaces will be attractive and encourage users to keep looking for new matches.

1.2.2. Reliability

- The application should be stable during events.
- The application should keep the entire matches and event histories.

1.2.3. Security / Confidentiality

- The user information such as previous matches, the events that the user attended will be secured from others.
- The application should not need the location information of a user except when they are attending an event.

1.2.4. Accessibility

- The application will be accessible to everyone with an Android or iOS mobile device and an internet connection.

- Any event organizer will be able to register their events in the application as long as they don't violate any rules.

1.2.5. Scalability

- The database management must be easily manageable in order to handle the increase in the user growth.
- The application should be able to handle a high number of users for each event room.
- Matching recommendation algorithm's performance should be scalable with respect to the number of all the users registered in the application.

1.2.6. Extensibility

- The system must be open to add new features in the future.
- The system should be easy to update with respect to both its background implementation and UI design

1.2.7. Performance / Efficiency

- The performance of the application depends on the active user number. By making predictions according to the event capacity, the possible user number could be forecasted.
- Matching recommendations must be processed in a reasonable amount of time so that the users will not wait for a noticeable time when they join an event.

1.3. Definitions, acronyms, and abbreviations

- **API**: Application Programming Interface.
- **REST** (or **REST API**): Representational state transfer, an architectural style used in the development of web services.
- **UI**: User Interface.
- **HTTP**: Hypertext Transfer Protocol.
- **Redux:** A widely used Javascript library for centralized state management.
- **TCP:** Transmission Control Protocol
- **SHA256:** A widely used cryptographic hash function.
- **QR Code:** Quick Response Code
- **BNF:** Backus-Naur Form
- **3NF:** Third Normal Form

1.4. Overview

In the rest of the report, the current software architecture will be briefly explained and then the proposed architecture will be explained in detail with a subsystem decomposition. Then, each part of the subsystem decomposition will be separately discussed. In the end, the details of how we worked as a team during the course will be explained.

2. Current software architecture

There are a lot of similar products in this area but the system/software architecture they offer is different from the architecture we propose since the requirements are different. In these architectures, all users are assumed to be in the same matching pool just restricted by their current geographic location.
In these applications, the terminology is swiping right or left to mean liking or skipping a profile as used in our application. When a user swipes right, a message is sent to the matchmaking service, preferably through a web socket, where the location manager determines which shard or matchmaking service this message should be sent to and routes it to the gateway, which connects to Kafka. Kafka is an open-source distributed event streaming platform, many platforms have been using Kafka in order to achieve high performance for event streaming [1]. These messages are then added to the queue. There will be one or more matchmaking services to which this information will be communicated, depending on the number of shards we have received as a consequence of the location manager service. Who is right shipping whom, where they are, and other metadata are all captured here.

There may be multiple workers reading messages from the Kafka queue concurrently. If A right swipes B, a record with the name "A B" is added to Redis and left alone. When B right swipes A, the match worker picks the message and checks in Redis whether "A has ever right-swiped B" i.e. we will definitely find key "A B" and check for metadata, which means a match has occurred, and the message will enter the matched queue, which is picked up by match notification and sent to both A and B via web socket saying "It's a match." What happens if A has never right swiped B for any reason? Then only the record "B A" will be added to Redis. When you right-swipe back B, it will check for the key before adding it [2].

3. Proposed software architecture

3.1. Subsystem decomposition

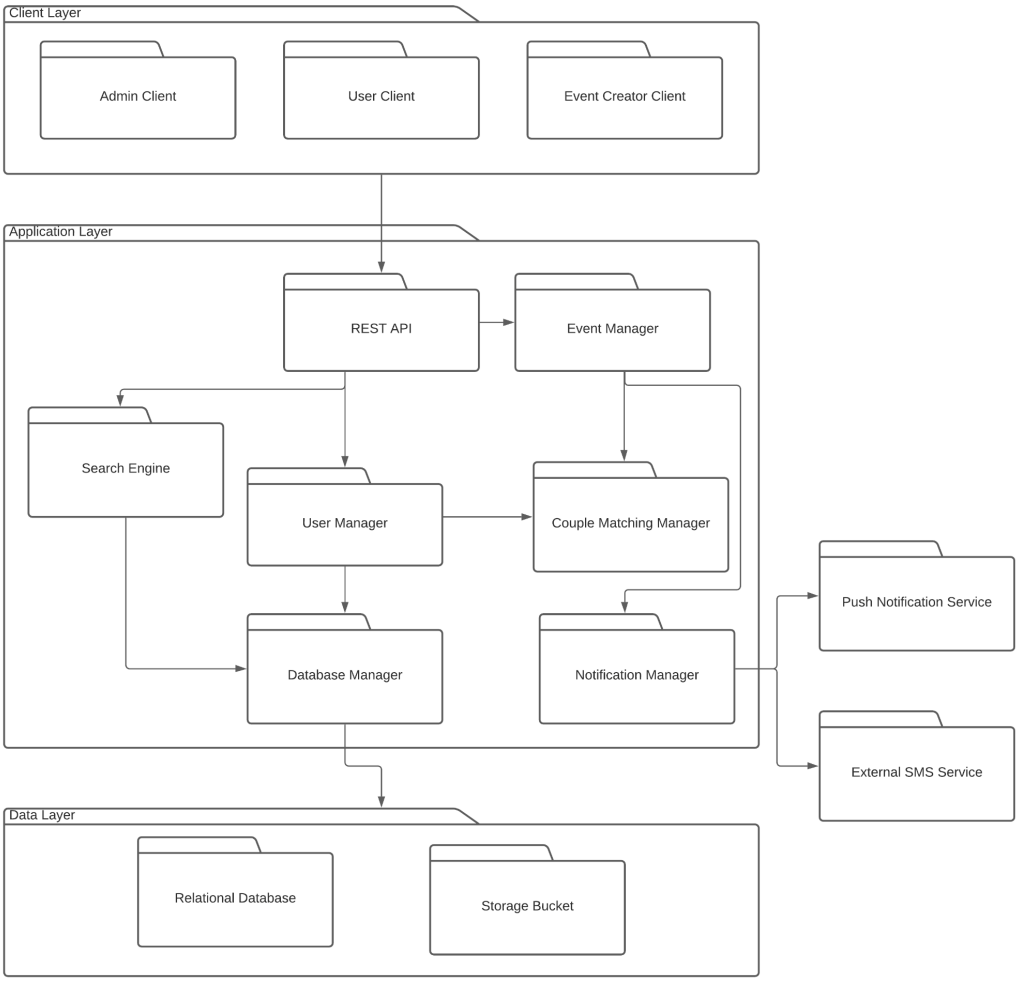

Figure 1. Subsystem Decomposition Diagram

3.2. Hardware/software mapping

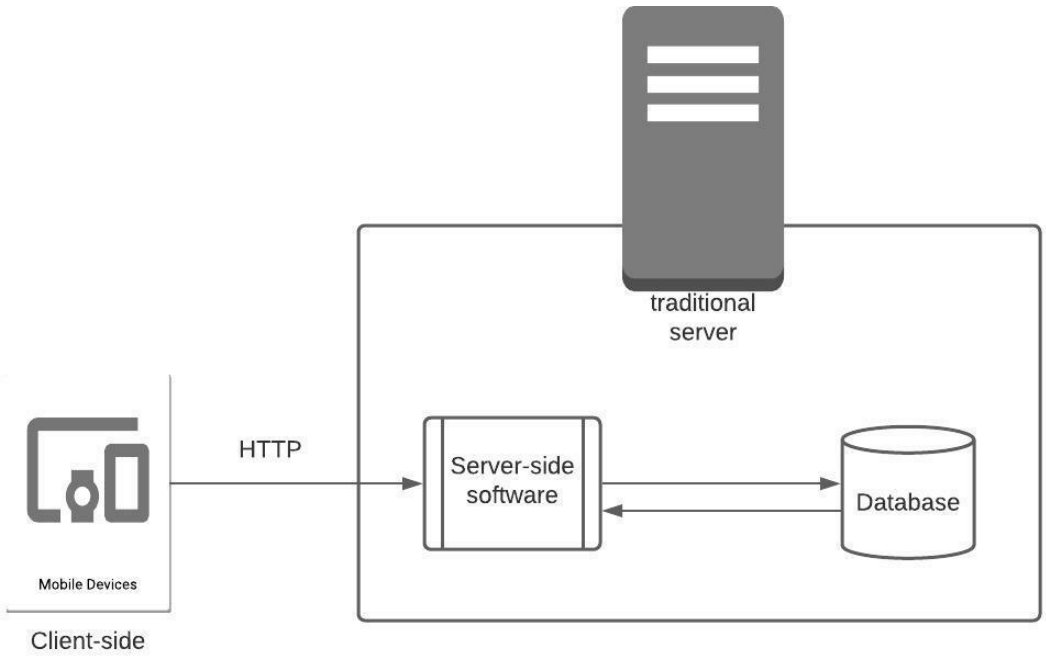

Figure 2. Deployment Diagram

The deployment will consist of the server-side and the client-side. The communication between these two parts will use HTTP requests from the client-side to the server-side. In the case of websocket usage, there can also be a full-duplex TCP communication channel between the client-side and the server-side. The server-side code will be deployed and run online classically. The client-side will be installed on either Android or iOS mobile devices by the users. The database will also be deployed together with the server-side and there will be no direct communication between the client-side and the database.

3.3. Persistent data management

Most of the subsystems rely on persistent data management in terms of responsiveness, reliability and extensibility. Data related to user profiles, events, event areas and matching between users will be stored in a relational database. PostgreSQL is chosen for this purpose because of its compatibility with the Django REST framework which will be used for building the Web API.

The database will be deployed on Heroku, along with the backend of the system to ensure fast response times for users and the security of the user data. Table schemes for the database will be designed as Views according to the Model-View-Template architecture used by Django. Separate from the relational database, user images will be stored in a cloud service. URLs for these images will be stored in the relational database, ready to be accessed with an API call.

Furthermore, local storage in mobile phones will also be used for persistent data management if applicable. Some data such as the chats between users might be persistently stored only locally. Extensions of redux libraries such as “redux-persist” [3], will be used in order to take advantage of the local storage in mobile phones.

3.4. Access control and security

Any action in the application will require a registered account. Registration and authentication will be accessible to everyone. But after the authentication, other API calls will require tokens that will be used to control the access levels. The public information of users such as their name, age, etc. will be visible to only other registered users that currently participate in the same event. The private data of users will be kept in a database and will be controlled with secure systems. The passwords of users will be hashed and secure login and authentication systems will be employed.

3.5. Global software control

Software control will be event driven. Client side will always be connected to the server and events happening on the client side will be handled by the server by calling the related service. Coupl’s core functionalities such as joining/leaving events and matching with other users are initiated by the user by them interacting with the application and the server handling the request.

In rare cases like sending event notifications, the action might be explicitly triggered by the server. Since the matching process will be interactive and there will also be a feature to use chat rooms, the actions of users affect each other. To provide this interactivity, websockets will be used. Websockets let the server notify the users efficiently by providing a full-duplex communication between the client and the server.

3.6. Boundary conditions

3.6.1. Initialization

To use the software, the user must have the application installed on their phones and must have an internet connection. If a QR code generated by our software is scanned through a mobile device, it will direct them to the appropriate page to install the application. For the first usage of the application on a mobile device, users must register a new account or log in with a previously created account. In the subsequent initializations of the application, the user will be directly logged in with their previous credentials.

3.6.2. Termination

If a user terminates the application while they are participating in an event, they will have ten minutes to start the application again and they will not be directly kicked from the event. After ten minutes of a connection loss, the user will leave the event automatically and the other event participants will not be able to see this user anymore. After leaving an event in any

way, the user might join the event again by scanning the same QR code and their actions from the previous session will not be lost.

3.6.3. Run-time

When a user loses their internet connection while they are participating in an event, the same rules for termination apply. In the case of a failure in the back-end, appropriate error messages will be shown in the application. Requests that result with a failure will not have any effect on the persistent data and these failures will be logged to be resolved later. Other types of failures are possible such as getting time-outs in the API requests. These may be caused by the poor internet connection of the user or a problem in the back-end servers, the user will be notified about this connection problem in both cases.

4. Subsystem services

4.1. Client Layer

4.1.1. Admin Client

Admin clients will be able to approve comments, event locations and events. Admin clients will also have access to the database and will be in charge of maintenance.

4.1.2. User Client

The user will be able to access the application from their mobile phones. These users denote the end-users.

4.1.3. Event Creator Client

Event creator clients will be able to add new locations, create events and check data about their previous events.

4.2. Application Layer

4.2.1. REST API

The gateway server which follows the REST API protocols will be a single entry point which will handle the requests coming from all of the clients. Authentication, authorization, encryption and decryption and session management will be handled by this server.

4.2.2. Couple Matching Manager

The Couple Matching Manager is the main part of our application. The matching recommendation algorithm will be working in this subsystem and it will show the possible matches to a user in the order of their relevance. When a user likes or skips another user in an event, this information will be stored by this manager. When there is a like from both sides, both users will be notified.

4.2.3. Notification Manager

The notification manager will send notification to upcoming events to the users using an external sms/email notification service. The users will be notified when the event they select as they are planning to join will start, they have a new match and they received a new message. On top of getting sms notifications they will also receive push notifications.

4.2.4. User Manager

This service is responsible for managing the end-users actions such as updating their profile, joining an event and selecting an event to join. To hold the appropriate data and fulfill the user's actions this service will be in communications with the database manager to update the database accordingly.

4.2.5. Search Engine

Search engine will manage in-app search functionalities. These functionalities will be:

- Users searching events by event name, tags and location.
- Event creators searching event places by place name and location.

4.2.6. Database Manager

Database manager will be a gateway, handling the data flow between the data layer and the services in the application layer.

4.2.7. Event Manager

Event manager is one of the most crucial subsystems, responsible for the logic behind the events. For event organizers, it will handle the addition of event locations to the system, organization of events in those locations and modification or deletion of the events. Moreover, it also manages the partition of event areas into subareas where users will meet. Additionally, the event manager communicates with the database through the database manager and provides event organizers statistics for their events. For the users, it will create virtual event rooms and will handle join/leave requests.

4.3. Data Layer

Data layer consists of a relational database and a storage bucket. Storage bucket will be a cloud server holding the user images and other larger scaled data. All the other data relating to users and events will be held in tables in the relational database. This includes the links to everything stored in the storage bucket. The tables will be designed in BNF or 3NF approach to prevent redundancy and ensure referential integrity.

5. Consideration of Various Factors in Engineering Design

5.1. Public Health

Coupl does not have a direct effect on public health. Unlike other dating apps, Coupl promotes socialization and real in-person dates. In this regard, it can be said that Coupl is a healthier option than its alternatives.

5.2. Public Safety

Coupl does not have a direct effect on public safety. On Couple, events are organized by event organizers or place owners. Couple only lists these events. Therefore, the main responsibility of holding safe events is up to the event organizers and place owners. However, Coupl is responsible for the safety of its users' data. To this end, we will use SHA256 to protect our user's passwords. We will use cross-site scripting protection, request forgery and injection protection. We will also use https and secure cookies to make sure our user's data is protected all of the time.

5.3. Public Welfare

Coupl does not affect public welfare.

5.4. Global Factors

Coupl can be used in any country. Therefore, to increase its usability and reach, it is important that Coupl supports English.
This is one of the most important factors that affected the design of the application.

5.5. Cultural Factors

On Coupl, different types of events can be held by event organizers and place owners. Since there is no restriction on the type of events that can be held, it increases cultural diversity and makes Coupl be in harmony with different cultures.

5.6. Social Factors

On Couple, any user who has the event QR code and is within the vicinity of the event place can attend the event room and try to match with other attendees regardless of their gender and sexual orientation. Since attendees can have different sexual orientations, it is important to take this into consideration in our implementation (matching algorithm).

5.7. Sustainability

There are no direct sustainability implications of the application. However, Coupl can be associated with socio-ecological sustainability through its social and cultural effects and since it interacts with the social networks by forming new relationships between individuals.

Factor	Effect Level	Effect
Public Health	3	A healthier design compared to the alternative applications.
Public Safety	8	The user data must be secured.
Public Welfare	0	-
Global Factors	8	English language option must be available.
Cultural Factors	1	Coupl is in harmony with different cultures.
Social Factors	7	All event participants should be able to use the application.
Sustainability	2	No direct effect, but already related to cultural and social factors.

Table 1. Effects of Various Factors

6. Teamwork Details

6.1. Contributing and functioning effectively on the team

We divided our group into two teams (frontend and backend) since it would be more effective and would help us move faster. Each team member primarily communicated with the members of the same team throughout the development. This helped us keep track of our own team's overall progress and made the development process more effective as we only had to talk with members of the same team instead of talking to the whole group and taking their time unnecessarily. Also, since each team member was kept up-to-date on their teammates' progress, it was easier to come up with the next tasks and assign them to the team members who were available.

Communication between these two teams was also necessary to define the API between frontend and backend. During our meetings with the whole team, we also discussed this and any other problem that relates to the whole project.

6.2. Creating a collaborative and inclusive environment

To create a collaborative and inclusive work environment, before starting the project, we held a meeting to talk about our interests and past experiences and discussed what part of the

project each group member should work on. This allowed us to divide the workload in a way that would match our interests and skills. Also, throughout the semester, we always stayed in touch with each other via Discord. To discuss the issues regarding the project such as the problems we faced during the implementation and how to solve them or what we should do next, we held meetings whenever it was necessary. We also held biweekly meetings to discuss the overall progress with the group. The discussions we had during these meetings allowed each group member to express their thoughts and helped us create a collaborative and inclusive environment.

6.3. Taking lead role and sharing leadership on the team

Our group did not have a designated leader. However, some group members took more responsibility in matters such as creating tasks and assigning them to available team members, setting up meetings and preparing report templates, which, in turn, helped ease the overall development process. In addition, each team regulated their own workload among themselves and each team member was responsible to the members of the same team. We also took advantage of the work packages we have created during the analysis of the project. When a decision was needed in any part of the project, the leader of the corresponding work package would take the lead role. Since the leaderships for these work packages were shared equally, this also shared the leadership throughout the project.

7. References

[1] *Documentation*. Apache Kafka. (n.d.). Retrieved December 23, 2021, from https://kafka.apache.org/documentation/
[2] srivastava, J. (2020, December 21). *Dating application system design*. Medium. Retrieved December 23, 2021, from https://medium.com/system-design-concepts/dating-application-system-design-aae411412267
[3] “Redux Persist, Persist and rehydrate a redux store.” Retrieved December 24, 2021. from https://www.npmjs.com/package/redux-persist