University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: 4

Date: 05/08/2024

Group Number: 38

Name	Student Number	Alias (Userid)	Preferred E-mail Address
Ali Hasan	34975292	ahasan02	ali.hasan9712@gmail.com
Sharjeel Shahid	30717987	sshahi03	sshahi28@uwo.ca
Muhammad Zaid Tahir	26857201	mtahir03	zaidt221325@gmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)
In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

University of British Columbia, Vancouver
Department of Computer Science

Repository Link:

https://github.students.cs.ubc.ca/CPSC304-2024S-T2/project_a9c5w_u4h4a_v9s6y

SQL Script:

Refer to repo, found in root/sql:
createSchemas.sql
insertSampleData.sql

Project Description:

Our full stack application accomplished the modeling and display of a player-centric client for an online, match-based, multiplayer video game. Using a combination of TypeScript, React, and postgres, we were able to create an application with player-persistent information. Specifically, centered around individual player profiles, we were able to display the player profile and match history/details. Further, the database was also able to model a centralized game shop supporting multiple currencies and item types, which can be used to populate player inventories. Players can interact with the intuitive and dynamic GUI elements to make these queries and subsequently observe the changes. The Toastify library provides error notifications. In addition, error handling and sanitation principles were observed through the use of try catch statements, curl testing, multiple levels of logging, and pgformat for SQL statements (avoid injections).

How Schema Differed from Milestone 3:
We had to add certain attributes we initially neglected to the Schemas. For example, we added datePlayed and outcome to match stats. Specifically, we believed these were valuable pieces of information the players would like to know and would be an important display.

Furthermore, although we had sample data in milestone 3, we chose to expand that data. Specifically, we increased the numbers of modes we support to show that our application is not statically reliant on matches only played in one mode. We also added more players, items, and examples in general. This was done to show the robustness of our application, and how it can deal with different scenarios. In short, it was done for future extensibility and updates we may have had. Additionally, these extra scenarios helped us test edge cases and the like (for example, how are matches of different modes displayed? What happens when items in inventory drop to 0?).

The final big change we made is that our initial SQL DDL statements had a double sided foreign key constraint on both the Match and MatchStats table. Initially we thought this was a good idea, however upon trying to alter match or match stats we found this was an oversight. When we tried to delete Matches, MatchStats would complain and vice versa. We also realized these constraints were too strict to be needed on both sides. We removed the constraint on the Matches sides, now instead choosing to join matchStats with the match table as a foreign key only.

University of British Columbia, Vancouver
Department of Computer Science

Copy of Schema/Screenshots of Data: Refer to Repo, specifically exampleSchema.pdf in misc/milestone4.

Queries:

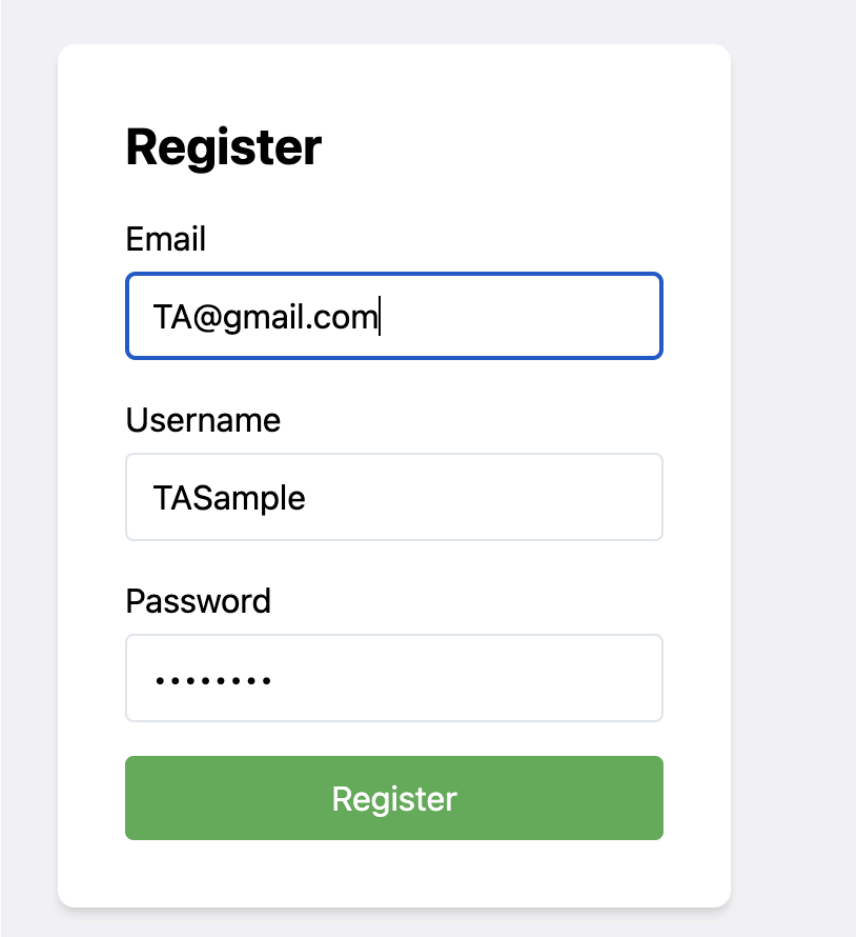
INSERT:
Located in `backend/src/controllers/users.ts" refer to function createUser:

```
 export async function createUser(
 name: string,
 email: string,
 password: string,
): Promise<any | null> {
 const sql = format(
   'INSERT INTO %I (username, email, password, elo, totalXP) VALUES (%L, %L,
%L, %L, %L) RETURNING playerID;',
   RELATION_NAME,
   name,
   email,
   password,
   0,
   0,
 );

 try {
   const rsp = await query(sql);
   if (rsp.rows && rsp.rows.length > 0) {
     return rsp.rows[0];
   } else {
     logger.error('User created but no playerID returned');
     return null;
   }
 } catch (err) {
   logger.error('Error executing query', err);
   return null;
 }
}
```

BEFORE:

```
zaidt=# SELECT * FROM Player;
 playerid | username |         email         |        password        | elo  | totalxp
----------+----------+-----------------------+------------------------+------+---------
        0 | admin    | admin@gamecompany.com | superSecurePassword123 | 9999 |  999999
        2 | Sharjeel | sharjeel@foo.com      | password1234567890     |  800 |    1830
        4 | Steve    | steve@foo.com         | askldfjdsl             | 1200 |    3840
        6 | Emma     | emma@foo.com          | securepass123          |  900 |    2850
        8 | Noah     | noah@foo.com          | noahsark789            | 1300 |    4320
       10 | Ava      | ava@foo.com           | avapwd987              | 1000 |    3060
        1 | Ali      | ali@foo.com           | password               | 1000 |   11925
        3 | Zaid     | zaid@foo.com          | qwerty                 |  700 |    6935
        5 | Wozniak  | wozniak@foo.com       | password1234567890     | 1550 |    5415
```

QUERY:



AFTER:

```
zaidt=# SELECT * FROM Player;
 playerid | username |         email         |        password        | elo  | totalxp
----------+----------+-----------------------+------------------------+------+---------
        0 | admin    | admin@gamecompany.com | superSecurePassword123 | 9999 |  999999
        2 | Sharjeel | sharjeel@foo.com      | password1234567890     |  800 |    1830
        4 | Steve    | steve@foo.com         | askldfjdsl             | 1200 |    3840
        6 | Emma     | emma@foo.com          | securepass123          |  900 |    2850
        8 | Noah     | noah@foo.com          | noahsark789            | 1300 |    4320
       10 | Ava      | ava@foo.com           | avapwd987              | 1000 |    3060
        1 | Ali      | ali@foo.com           | password               | 1000 |   11925
        3 | Zaid     | zaid@foo.com          | qwerty                 |  700 |    6935
        5 | Wozniak  | wozniak@foo.com       | password1234567890     | 1550 |    5415
        7 | Olivia   | olivia@foo.com        | oliviapass456          | 1100 |    3445
        9 | Liam     | liam@foo.com          | liamstrong321          | 1400 |    4955
       11 | TASample | TA@gmail.com          | tasample               |    0 |       0
```

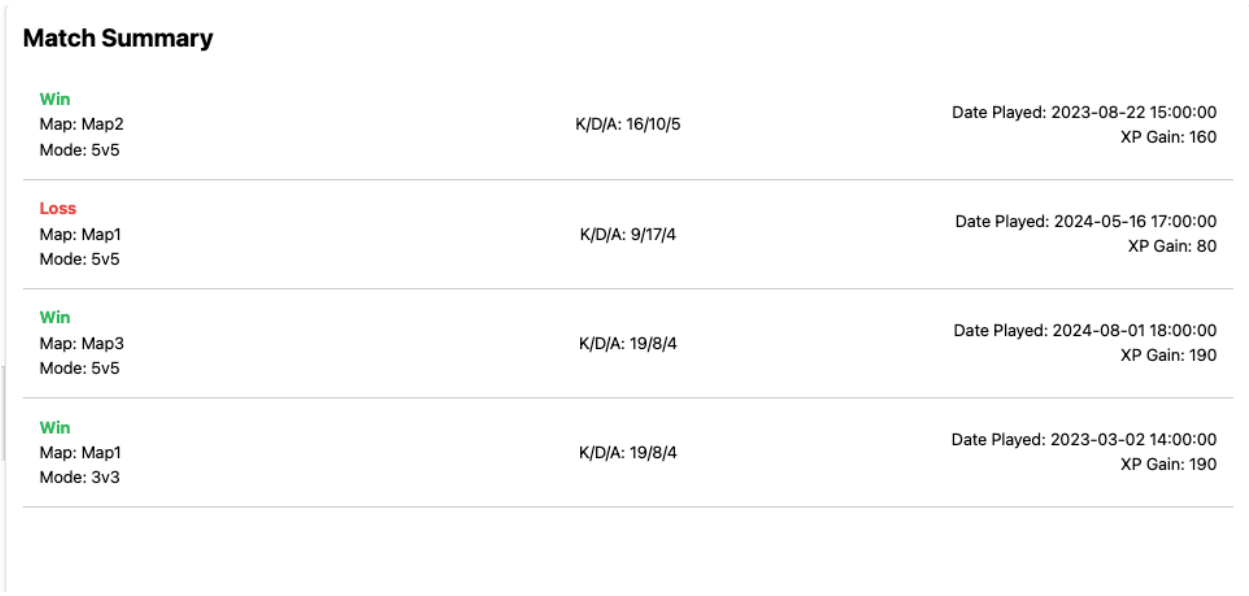
DELETE:

Located in `backend/src/controllers/match.ts" refer to function deleteMatch

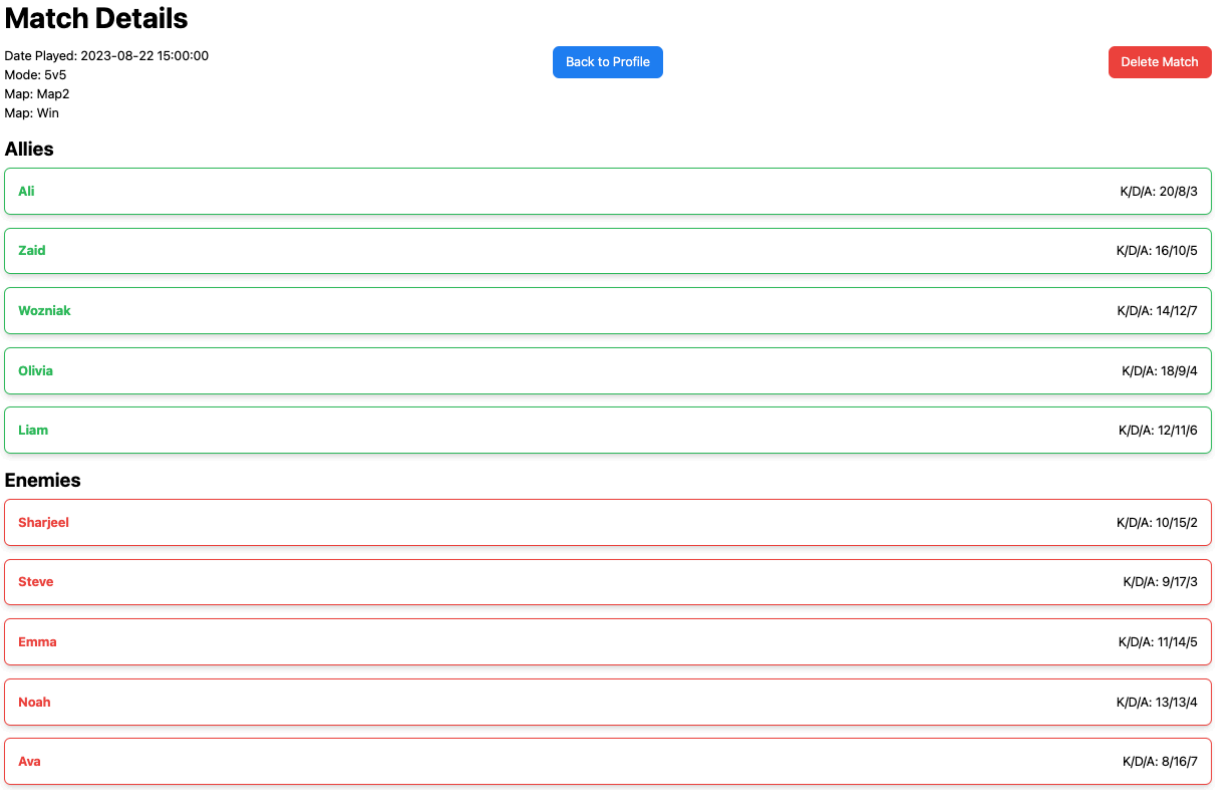
```
export async function updatePlayerXP(playerID: number, xpGain: number):
Promise<boolean> {
const updatePlayerSql = format(
   `UPDATE Player
    SET totalXP = totalXP - %L
    WHERE playerID = %L;`,
   xpGain,
   playerID);

 try {
   await query(updatePlayerSql);
   return true;
 } catch (err) {
   logger.error('Error updating player XP', err);
   return false;
 }
}

export async function deleteMatch(matchID: number): Promise<boolean> {
 const getPlayerXpSQL = format(
   `SELECT playerID, xpGain
    FROM MatchStats
    WHERE matchID = %L;`,
   matchID);

 const deleteMatchSQL = format(
   `DELETE FROM Match
    WHERE matchID = %L;`,
   matchID);
```

```
try {
  const playerXpRsp = await query(getPlayerXpSQL);
  if (playerXpRsp.rows && playerXpRsp.rows.length > 0) {
    for (const row of playerXpRsp.rows) {
      const success = await updatePlayerXP(row.playerid, row.xpgain);
      if (!success) {
        throw new Error('Failed to update player XP');
      }
    }
  } else {
    logger.error('No players found for the match');
    return false;
  }
  await query(deleteMatchSQL);

  return true;
} catch (err) {
  logger.error('Error deleting match', err);
  return false;
}
}
```

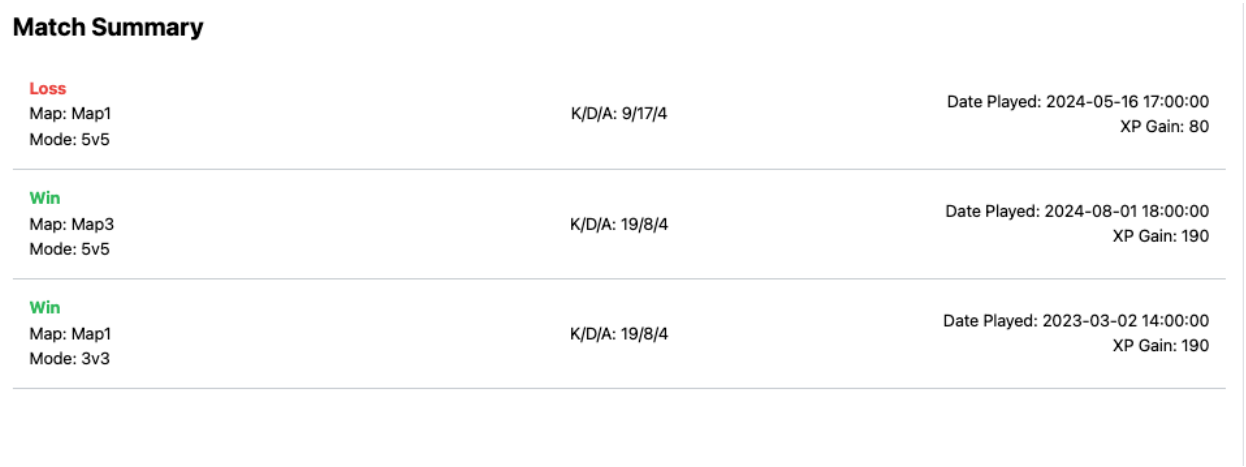
RESULT:



Delete match 1 (ie. map 2 certain K/D/A)



It doesn't appear anymore, and is deleted as well as the match stats are deleted and xp Updated/lost.



University of British Columbia, Vancouver
Department of Computer Science

UPDATE:
Located in `backend/src/controllers/users.ts" refer to function updateUserByID

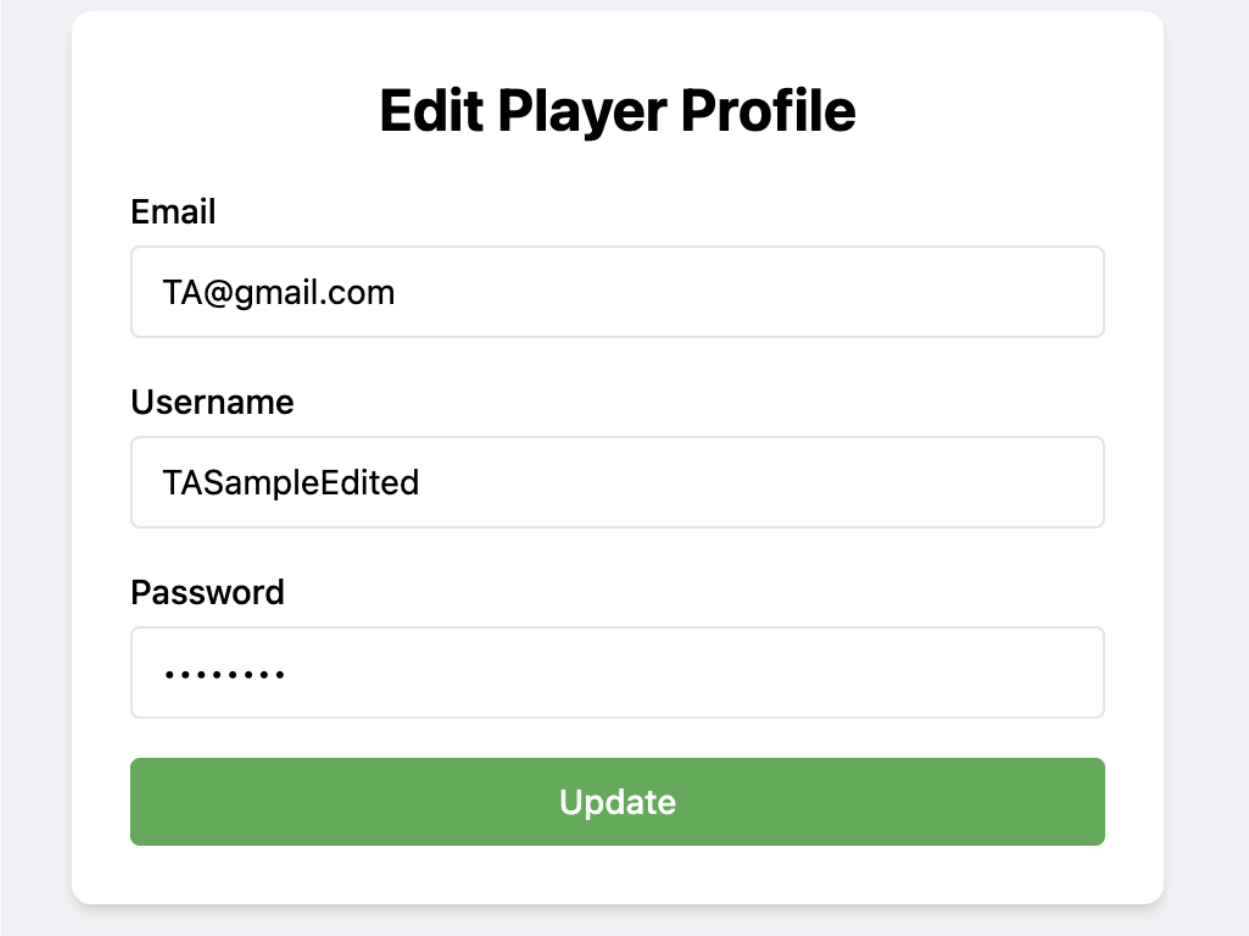
```
export async function updateUserByID(
 playerID: number,
 values: Record<string, string | number>,
): Promise<boolean> {
 const setClause = Object.entries(values)
   .map(([key, value]) => {
     if (typeof value === 'number') {
       value = Math.floor(value);
     }
     return format('%I = %L', key, value);
   })
   .join(', ');

 const sql = format(
   'UPDATE %I SET %s WHERE playerID = %L;',
   RELATION_NAME,
   setClause,
   playerID,
 );

 try {
   await query(sql);
   return true;
 } catch (err) {
   logger.error('Error updating user', err);
   return false;
 }
}
```

University of British Columbia, Vancouver
Department of Computer Science

BEFORE:

```
zaidt=# SELECT * FROM Player;
 playerid | username |         email         |        password        | elo  | totalxp
----------+----------+-----------------------+------------------------+------+---------
        0 | admin    | admin@gamecompany.com | superSecurePassword123 | 9999 |  999999
        2 | Sharjeel | sharjeel@foo.com      | password1234567890     |  800 |    1830
        4 | Steve    | steve@foo.com         | askldfjdsl             | 1200 |    3840
        6 | Emma     | emma@foo.com          | securepass123          |  900 |    2850
        8 | Noah     | noah@foo.com          | noahsark789            | 1300 |    4320
       10 | Ava      | ava@foo.com           | avapwd987              | 1000 |    3060
        1 | Ali      | ali@foo.com           | password               | 1000 |   11925
        3 | Zaid     | zaid@foo.com          | qwerty                 |  700 |    6935
        5 | Wozniak  | wozniak@foo.com       | password1234567890     | 1550 |    5415
        7 | Olivia   | olivia@foo.com        | oliviapass456          | 1100 |    3445
        9 | Liam     | liam@foo.com          | liamstrong321          | 1400 |    4955
       11 | TASample | TA@gmail.com          | tasample               |    0 |       0
```

QUERY:



AFTER:

playerid	username	email	password	elo	totalxp
0	admin	admin@gamecompany.com	superSecurePassword123	9999	999999
2	Sharjeel	sharjeel@foo.com	password1234567890	800	1830
4	Steve	steve@foo.com	askldfjdsl	1200	3840
6	Emma	emma@foo.com	securepass123	900	2850
8	Noah	noah@foo.com	noahsark789	1300	4320
10	Ava	ava@foo.com	avapwd987	1000	3060
1	Ali	ali@foo.com	password	1000	11925
3	Zaid	zaid@foo.com	qwerty	700	6935
5	Wozniak	wozniak@foo.com	password1234567890	1550	5415
7	Olivia	olivia@foo.com	oliviapass456	1100	3445
9	Liam	liam@foo.com	liamstrong321	1400	4955
11	TASampleEdited	TA@gmail.com	tasample	0	0

SELECTION:
Located in `backend/src/controllers/match.ts" refer to function getPlayerMatches

```
export async function getPlayerMatches(playerID: number): Promise<any | null>
{
 const sql = format(
   `SELECT
     m.matchID,
     m.mode,
     m.map,
     m.datePlayed,
     ms.xpGain,
     ms.kills,
     ms.deaths,
     ms.assists,
     ms.outcome
   FROM
     Match m
   JOIN
     MatchStats ms
   ON
     m.matchID = ms.matchID
   WHERE
     ms.playerID = %L;`,
   playerID);

 try {
   const rsp = await query(sql);
   if (rsp.rows && rsp.rows.length > 0) {
     return rsp.rows;
   } else {
     logger.error('No matches found for player');
     return null;
```

```
    }
  } catch (err) {
    logger.error('Error executing query', err);
    return null;
  }
}
```

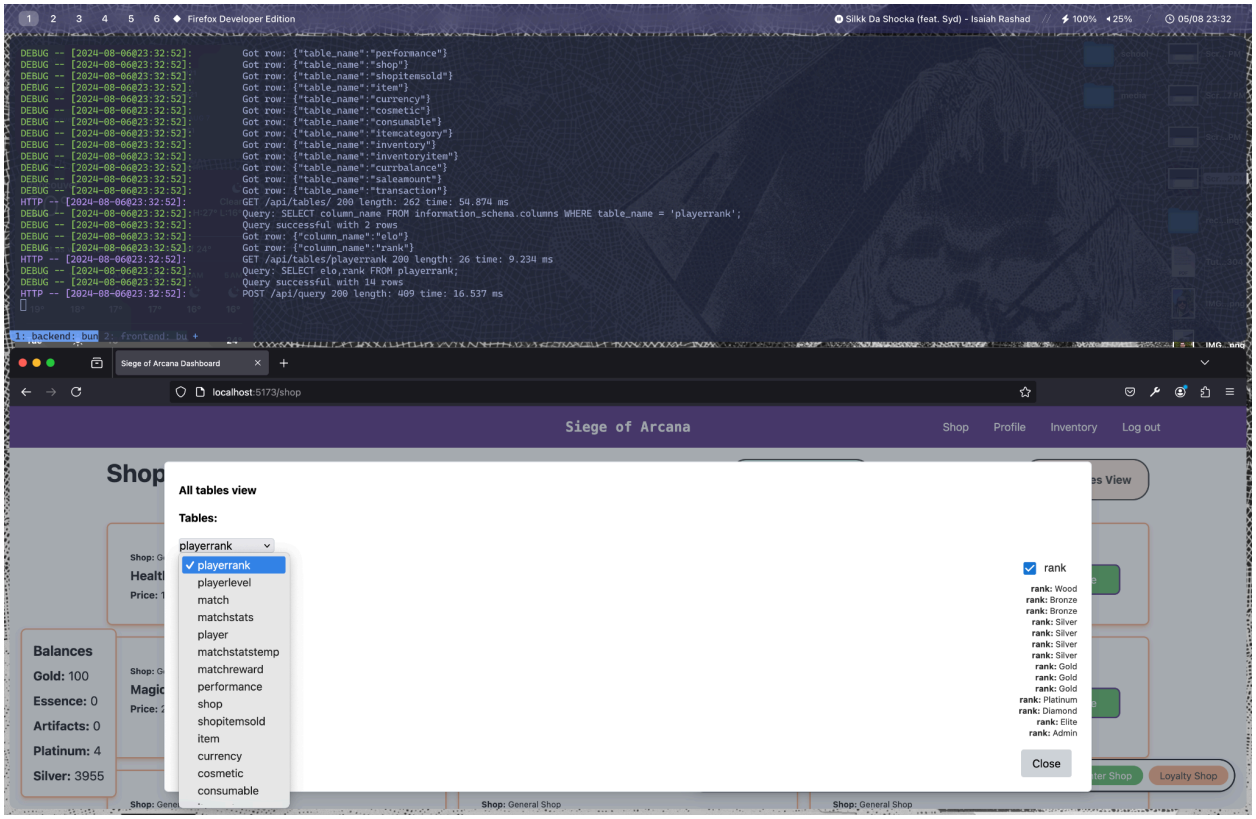
RESULT: (ie. on a particular profile this is automatically done when going to profile page, matchSummary widget).

PROJECTION:
Located in `backend/src/controllers/meta.ts`, refer to functions getColNames (to populate column names dynamically), getRelations (to populate relations dynamically), and projectSelect (to actually do the query).

```
// get all columns from a relation
1+ usages  UBC Student
export async function getColumnNames(relation: string): Promise<string[]> {
  const sql = format(
    'SELECT column_name FROM information_schema.columns WHERE table_name = %L;',
    relation,
  );
  const rsp = await query(sql);
  const retArr: string[] = [];
  rsp.rows.forEach(row => {
    logger.debug(`Got row: ${JSON.stringify(row)}`);
    retArr.push(row.column_name);
  });
  return retArr;
}

// get all relation names
1+ usages  UBC Student
export async function getRelations(): Promise<string[]> {
  const sql = format(
    'SELECT table_name FROM information_schema.tables WHERE table_schema = %L;',
    'public',
  );
  const rsp = await query(sql);
  const retArr: string[] = [];
  rsp.rows.forEach(row => {
    logger.debug(`Got row: ${JSON.stringify(row)}`);
    retArr.push(row.table_name);
  });
  return retArr;
}

1+ usages  UBC Student
export async function projectSelect(
  relation: string,
  columns: string[],
): Promise<object[]> {
  const sql = format('SELECT %I FROM %I;', columns, relation);
  const rsp = await query(sql);
  if (!rsp.rows) {
    logger.warn('No rows returned from query');
  }
  return rsp.rows;
}
```

University of British Columbia, Vancouver

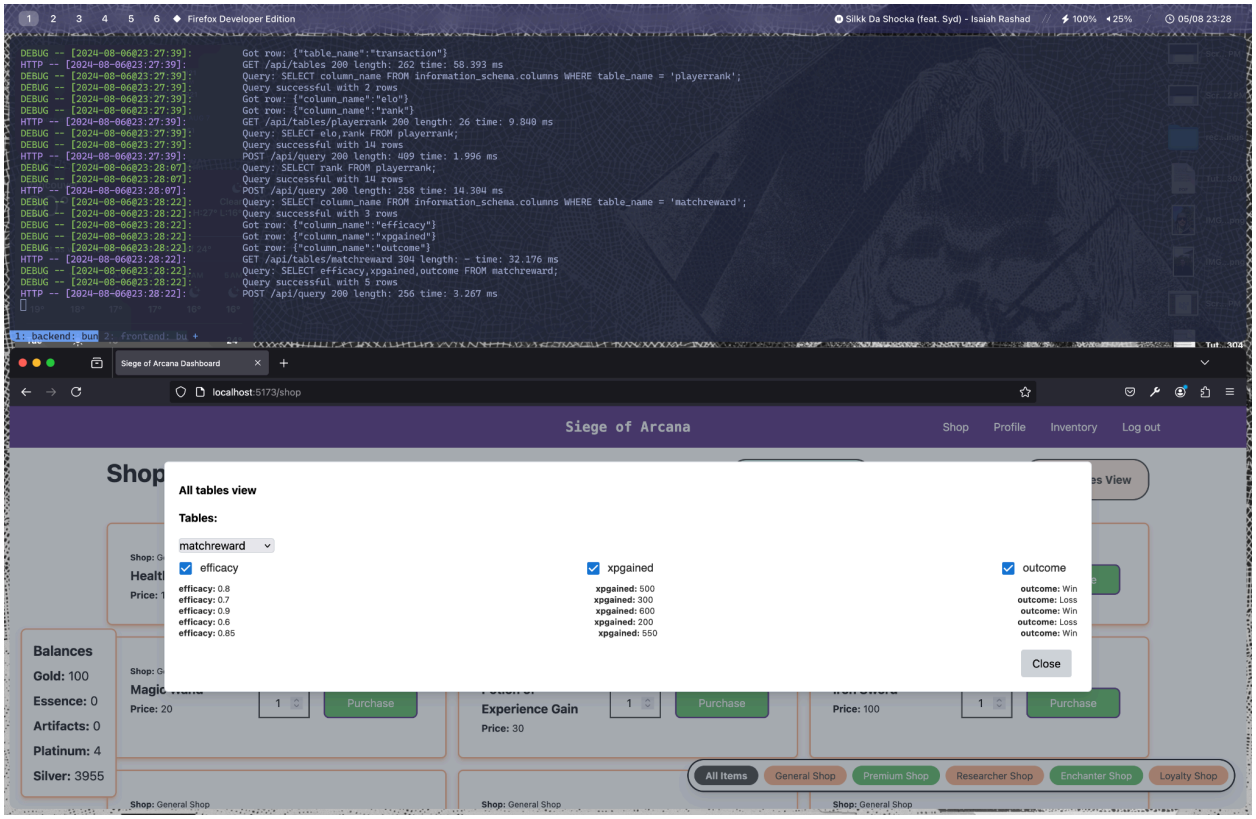
Department of Computer Science



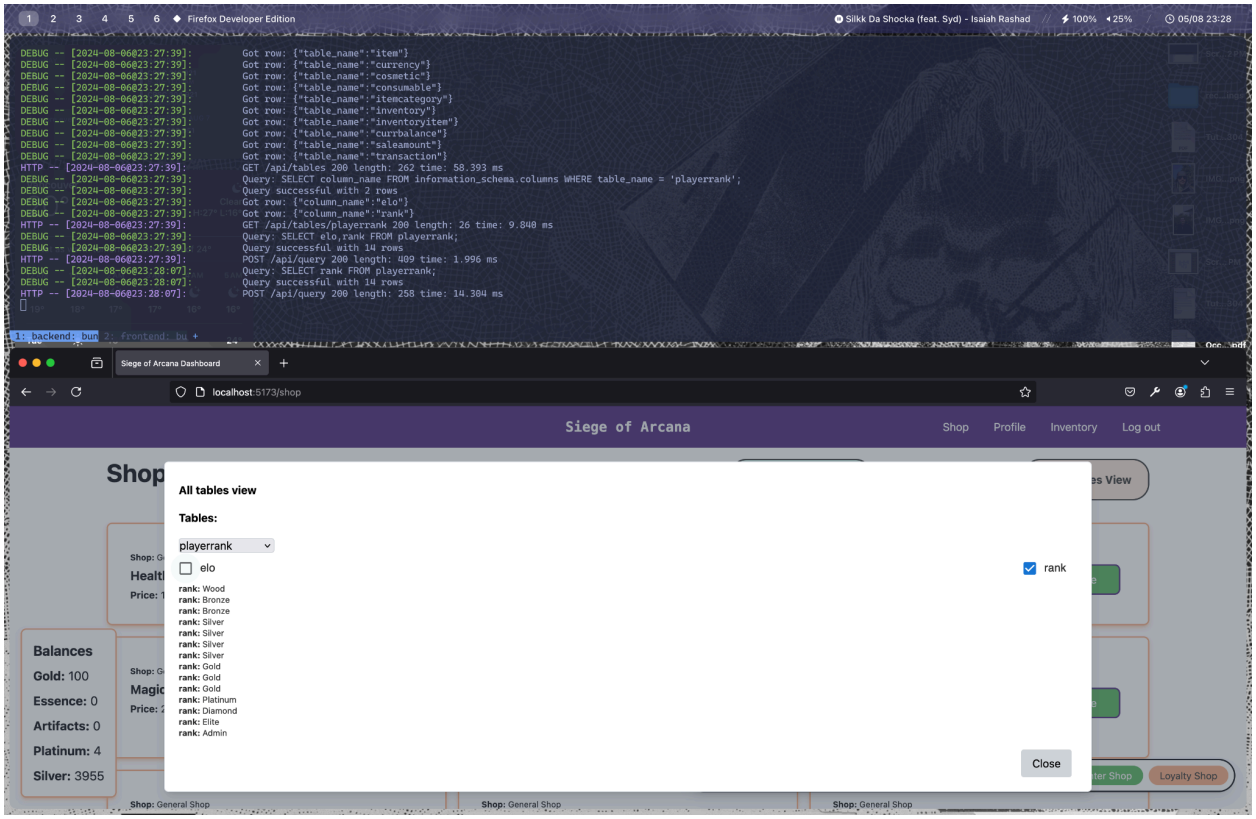
Dynamic population of table names, with log messages

University of British Columbia, Vancouver

Department of Computer Science



Changing views changes the data shown by calling a new query (see log messages)



Changing selected columns projects the data shown by calling a new query (see log messages)

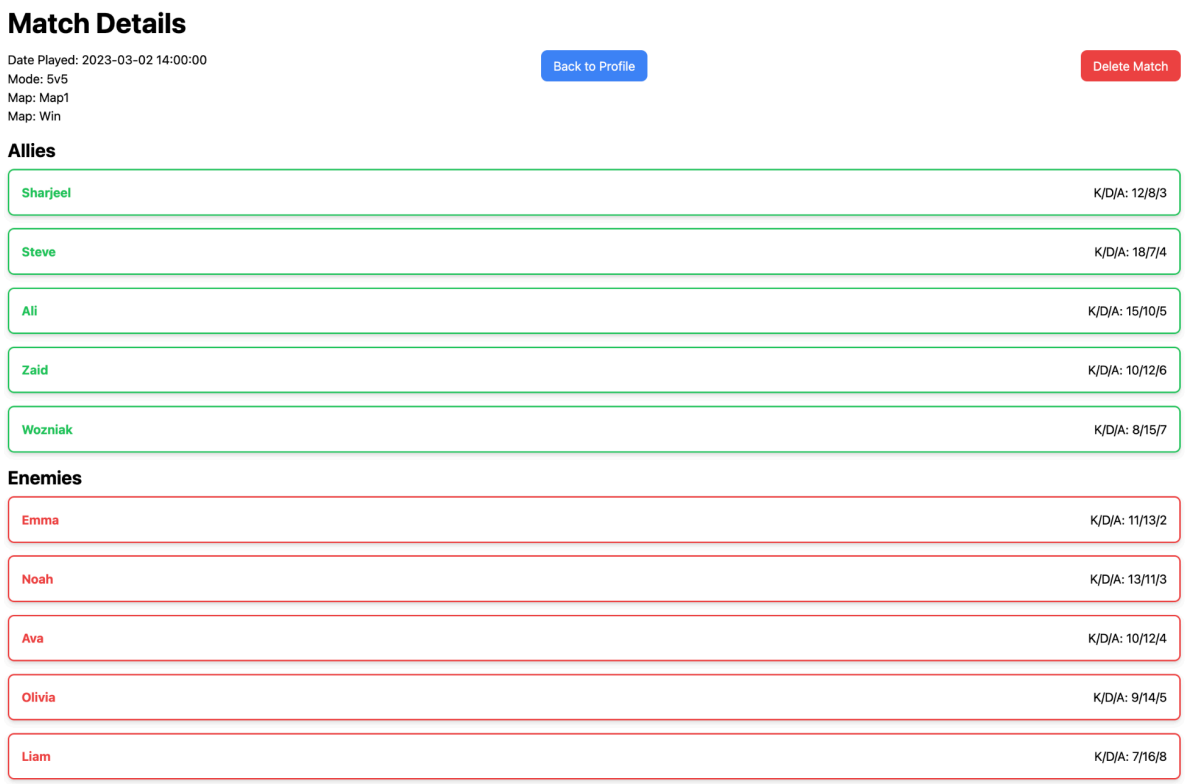
JOIN:
Located in `backend/src/controllers/match.ts" refer to function getMatchStats

```
export async function getMatchStats(matchIDS: number, playerIDS: number):
Promise<MatchView | null> {
 const sql = format(`
   SELECT
     m.matchID,
     m.mode,
     m.map,
     m.dateplayed,
     p.username AS playername,
     ms.xpGain,
     ms.kills,
     ms.deaths,
     ms.assists,
     ms.outcome
   FROM
     Match m
   JOIN
     MatchStats ms ON m.matchID = ms.matchID
   JOIN
```

```
    Player p ON ms.playerID = p.playerID
  WHERE
    m.matchID = %L;
`, matchIDS);

const out = format(`
  SELECT
    ms.outcome
  FROM
    Match m
  JOIN
    MatchStats ms ON m.matchID = ms.matchID
  JOIN
    Player p ON ms.playerID = p.playerID
  WHERE
    m.matchID = %L and p.playerID = %L
`, matchIDS, playerIDS);

try {
  const ally = await query(out);
  const rsp = await query(sql);
  if (rsp.rows && rsp.rows.length > 0 && ally.rows && ally.rows.length > 0) {
    const { matchID, mode, map, dateplayed } = rsp.rows[0];
    const outcome = ally.rows[0].outcome;

    const allies: Player[] = [];
    const enemies: Player[] = [];

    rsp.rows.forEach(row => {
      const player = {
        profile: row.playername,
        kda: `${row.kills}/${row.deaths}/${row.assists}`,
        efficacy: null,
      };
      if (row.outcome === outcome) {
        allies.push(player);
      } else {
        enemies.push(player);
      }
    });

    return {
      matchID,
      dateplayed,
      mode,
      map,
      outcome,
      allies,
      enemies,
    };
  } else {
    logger.error('No matches found for player');
    return null;
  }
```

```
 } catch (err) {
   logger.error('Error executing query', err);
   return null;
 }
}
```

RESULT: (ie. when clicking a match from match widget summary we arrive here we must get match stats by joining on a match, this is the view from clicking on the first match above).



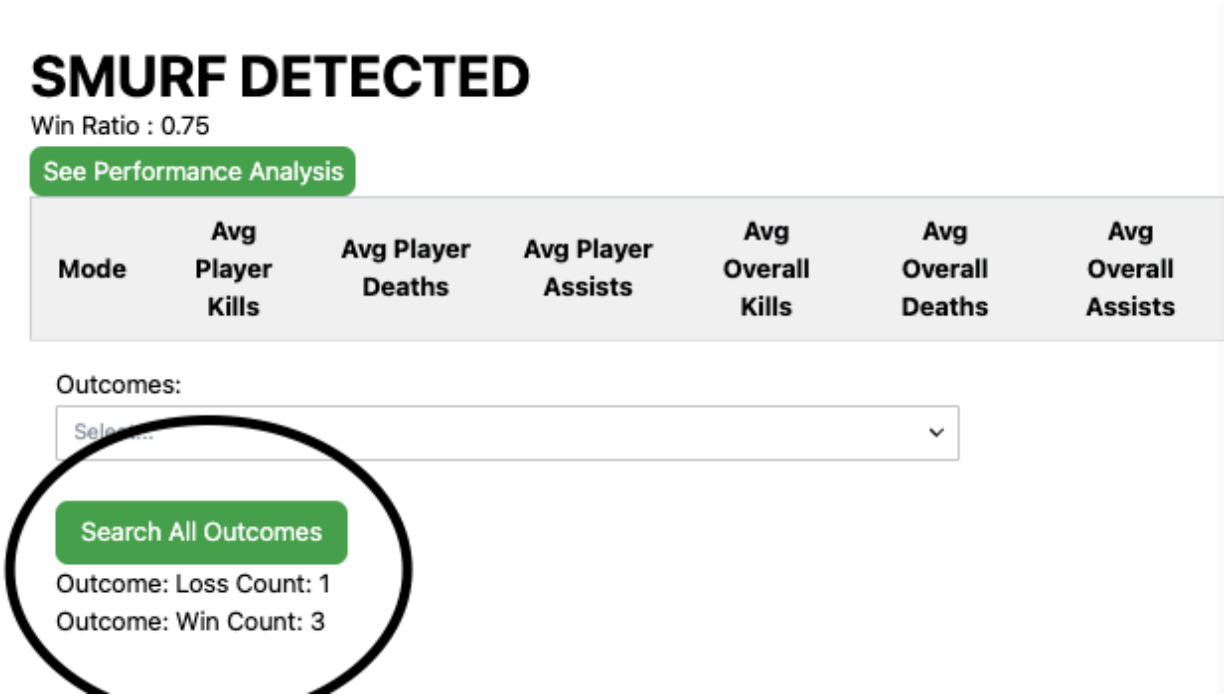
AGGREGATION WITH GROUP BY:

Located in `backend/src/controllers/match.ts" refer to function getPlayerOutcome

```
export async function getPlayerOutcome( playerID: number,
 outcome: string,
): Promise<object[] | null> {
 let sql;
 if (outcome === 'all') {
   sql = format(
     `
     SELECT
       playerID,
       outcome,
       COUNT(outcome)
     FROM
       %I
     WHERE
       playerID = %L
```

```
    GROUP BY
      playerID, outcome;
    `,
    STATS_RELATION,
    playerID,
  );
} else {
  sql = format(
    `
    SELECT
      playerID,
      outcome,
      COUNT(outcome)
    FROM
      %I
    WHERE
      playerID = %L
      AND outcome = %L
    GROUP BY
      playerID, outcome;
    `,
    STATS_RELATION,
    playerID,
    outcome,
  );
}

const rsp = await query(sql);
if (queryIsEmpty(rsp)) {
  logger.warn(
    `No matches found for playerID: ${playerID} with outcome: ${outcome}`,
  );
  return null;
}

return rsp.rows;
}
```

RESULT: Loss count/win count.

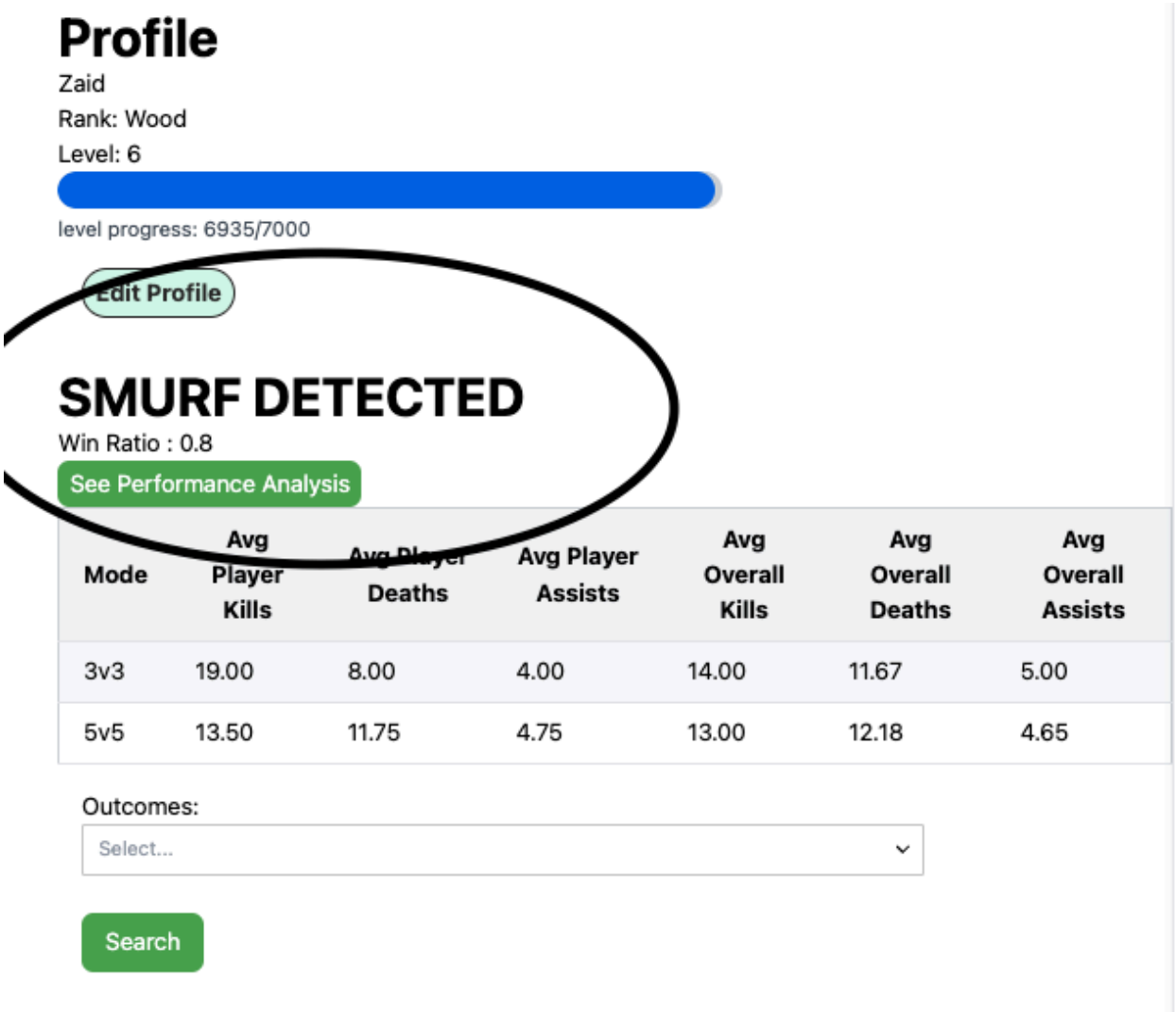


AGGREGATION WITH HAVING:
Located in `backend/src/controllers/match.ts" refer to function getPlayerIsSmurf

```
export async function getPlayerIsSmurf(
 playerID: number,
): Promise<{ isSmurf: boolean; winRatio: number } | null> {
 // determine if a player has 60% winrate using HAVING
 const sql = format(
   `
SELECT
   playerID,
   CAST(COUNT(CASE WHEN outcome = %L THEN 1 END) AS FLOAT) / NULLIF(COUNT(CASE
WHEN outcome IN (%L, %L) THEN 1 END), 0) as win_ratio
FROM
   %I
WHERE
   playerID = %L
GROUP BY
   playerID
HAVING
   CAST(COUNT(CASE WHEN outcome = %L THEN 1 END) AS FLOAT) / NULLIF(COUNT(CASE
WHEN outcome IN (%L, %L) THEN 1 END), 0) > %L
ORDER BY
   win_ratio DESC;
 `,
   WIN_STR,
   WIN_STR,
   LOSS_STR,
```

```
  STATS_RELATION,
  playerID,
  WIN_STR,
  WIN_STR,
  LOSS_STR,
  SMURF_THRESHOLD,
);

const rsp = await query(sql);
if (queryIsEmpty(rsp)) {
  return { isSmurf: false, winRatio: 0 };
}
return { isSmurf: !queryIsEmpty(rsp), winRatio: rsp.rows[0]?.win_ratio };
}
```

Result: Smurf detected if winrate above threshold.



University of British Columbia, Vancouver

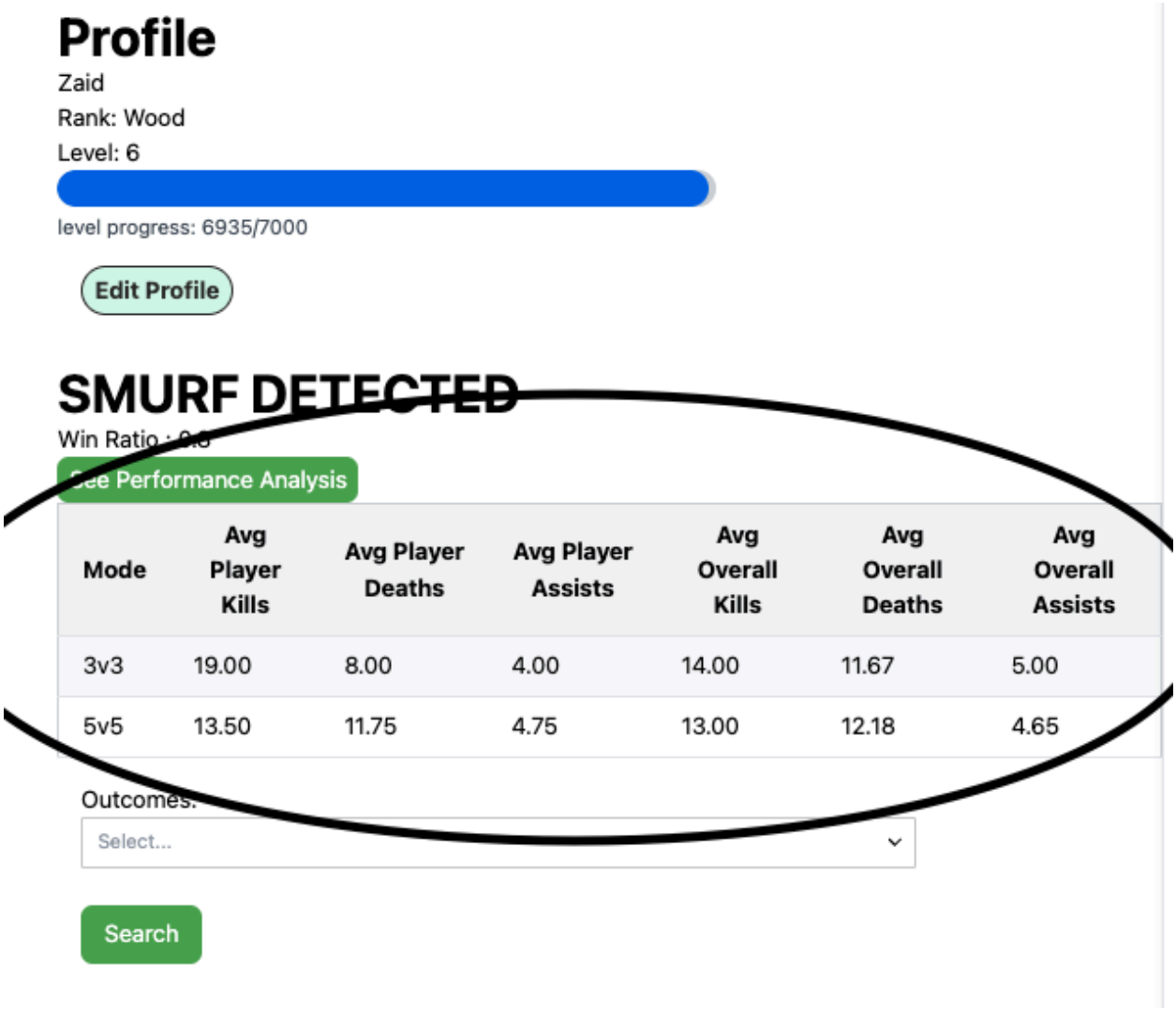
Department of Computer Science

NESTED AGGREGATION WITH GROUP BY:

Located in `backend/src/controllers/user.ts" refer to function getPlayerPerformanceAnalysis

```
export async function getPlayerPerformanceAnalysis(playerID: number):
Promise<any> {
const sql = format(`
   WITH OverallAverages AS (
     SELECT
       m.mode,
       AVG(ms.kills) as avg_overall_kills,
       AVG(ms.deaths) as avg_overall_deaths,
       AVG(ms.assists) as avg_overall_assists
     FROM MatchStats ms
     JOIN Match m ON ms.matchID = m.matchID
     GROUP BY m.mode
   )
   SELECT
     m.mode,
     AVG(ms.kills) as avg_player_kills,
     AVG(ms.deaths) as avg_player_deaths,
     AVG(ms.assists) as avg_player_assists,
     oa.avg_overall_kills,
     oa.avg_overall_deaths,
     oa.avg_overall_assists
   FROM MatchStats ms
   JOIN Match m ON ms.matchID = m.matchID
   JOIN OverallAverages oa ON m.mode = oa.mode
   WHERE ms.playerID = %L
   GROUP BY m.mode, oa.avg_overall_kills, oa.avg_overall_deaths,
oa.avg_overall_assists
   ORDER BY m.mode;
 `,
 playerID);

 try {
   const rsp = await query(sql);
   if (rsp.rows && rsp.rows.length > 0) {
     return rsp.rows;
   } else {
     logger.error('User created but no playerID returned');
     return null;
   }
 } catch (err) {
   logger.error('Error executing query', err);
   return null;
 }

}
```

Result: Performance Analysis for a particular profile



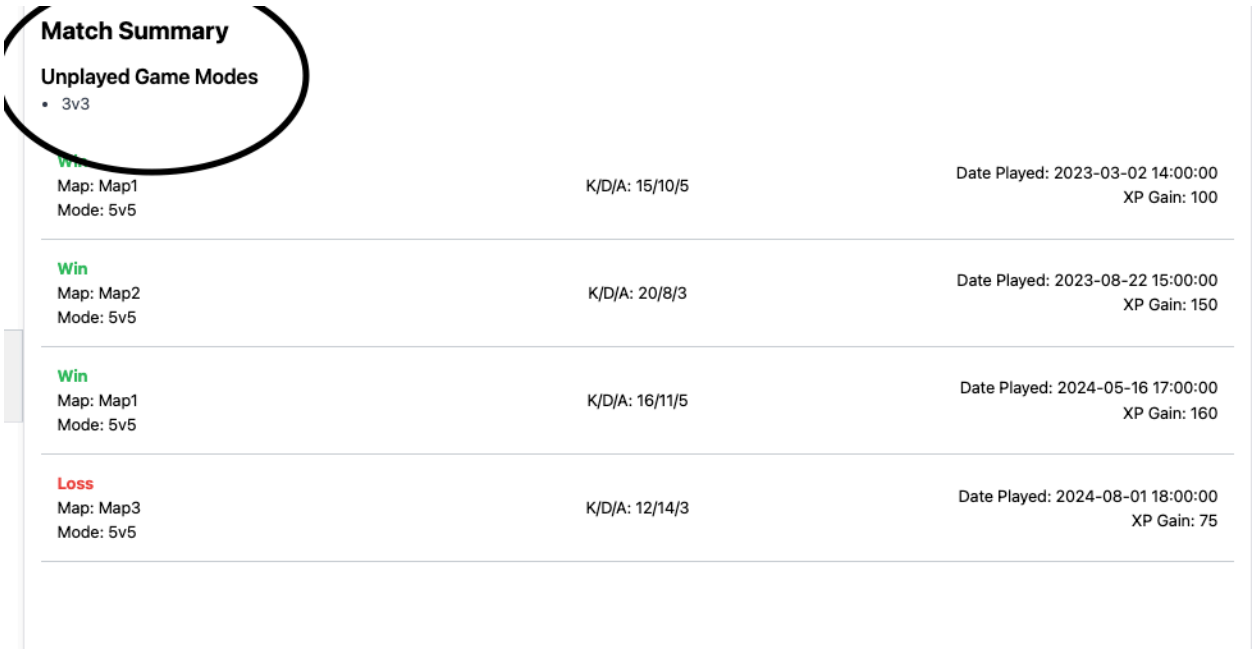
DIVISION:

Located in `backend/src/controllers/match.ts" refer to function getGameModesNotPlayed

```
export async function getGameModesNotPlayed(playerID: number):
Promise<string[]> {
const sql = format(
   `WITH AllGameModes AS (
       SELECT DISTINCT mode
       FROM Match
   ),
   PlayerGameModes AS (
       SELECT DISTINCT m.mode
       FROM MatchStats ms
       JOIN Match m ON ms.matchID = m.matchID
       WHERE ms.playerID = %s
   )
```

```
    SELECT agm.mode AS unplayedmode
    FROM AllGameModes agm
    WHERE NOT EXISTS (
        SELECT 1
        FROM PlayerGameModes pgm
        WHERE pgm.mode = agm.mode
    );`,
    playerID);

  try {
    logger.debug('playerID:', playerID);
    const result = await query(sql);
    logger.debug('Query result:', result.rows);
    return result.rows.map(row => row.unplayedmode);
  } catch (err) {
    logger.error('Error fetching game modes not played', err);
    return [];
  }
}
```

Result: Displays game modes not played by that user on top of match summary widget.



README.txt: in main project directory/github as well as MISC.